TURING 图灵程序设计丛书

[PACKT] PUBLISHING



[美] Megan Squire 著　任政委 译

干净的数据

数据清洗入门与实践

Clean Data

掌握高效数据清洗方法，为数据挖掘提供便利，让用户更好地体验大数据价值！

中国工信出版集团　人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

作 者 简 介



Megan Squire

依隆大学计算科学专业教授，主要教授数据库系统、Web开发、数据挖掘和数据科学课程。有二十年的数据收集与清洗经验。她还是FLOSSmole研究项目的领导者，致力于收集与分析数据，以便研究免费软件、自由软件和开源软件的开发。

译 者 简 介



任政委

辽宁滨城大连现役程序员一枚，长期从事一线软件开发工作，近年来为成为一名“思路清晰”“视角独特”“不搞办公室政治”“输出有生命力代码”“凭借技术知识普惠初中级IT从业者”的终身制全栈式程序员而不懈努力。曾经翻译《Oracle PL/SQL攻略》一书，并希望这本《干净的数据》能够为奋战在IT前线上的各界小伙伴们带来日常工作之外的另类体验。

TURING 图灵程序设计丛书



[美] Megan Squire 著　任政委 译

干净的数据

数据清洗入门与实践

Clean Data

人 民 邮 电 出 版 社
北　京

图书在版编目（CIP）数据

干净的数据 ：数据清洗入门与实践 / （美）斯夸尔
(Squire,M.）著 ；任政委译. -- 北京 ：人民邮电出版
社，2016.5
（图灵程序设计丛书）
ISBN 978-7-115-42047-3

Ⅰ. ①干… Ⅱ. ①斯… ②任… Ⅲ. ①数据处理
Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2016)第062910号

内 容 提 要

本书主要内容包括：数据清洗在数据科学领域中的重要作用，文件格式、数据类型、字符编码的基本概念，组织和处理数据的电子表格与文本编辑器，各种格式数据的转换方法，解析和清洗网页上的 HTML 文件的三种策略，提取和清洗 PDF 文件中数据的方法，检测和清除 RDBMS 中的坏数据的解决方案，以及使用书中介绍的方法清洗来自 Twitter 和 Stack Overflow 的数据。

本书适合任何水平的数据科学家以及对数据清理感兴趣的读者阅读。

◆ 著　　　　[美] Megan Squire
译　　　　任政委
责任编辑　岳新欣
执行编辑　李　敏
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：800×1000　1/16
印张：12.5
字数：296千字　　　　2016年 5 月第 1 版
印数：1 - 3 000册　　　2016年 5 月北京第 1 次印刷
著作权合同登记号　图字：01-2015-7995号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广字第 8052 号

版权声明

前　言

“请问，巴贝奇先生，如果把错误的数据放进机器中，是否能够得到正确的答案呢？”

——查尔斯·巴贝奇（1864）

“错进，错出。”

——美国国税局（1963）

“压根儿就没有干净的数据集。”

——乔希·沙利文，《财富》杂志收录的博思艾伦咨询公司副总裁语录（2015）

世界上第一台计算机的发明者查尔斯·巴贝奇，在他1864年的随笔文集中记录了这样一件事，他曾经因为有人认为在输入错误数据的情况下计算机依然能够给出正确答案而错愕不止。100年以后，美国税务部门开始耐心地向人们解释“错进，错出”，以此来表达即便是能力再强的税收官，在用计算机处理数据时，依旧要依赖输入数据的质量。又过了50年，到了2015年，在这个看起来超级神奇的时代，有着机器学习、自动纠错、预想接口以及比我们本人更为了解我们自己的各种推荐系统。然而，这一切的算法背后，仍旧需要高质量的数据来保证学习的正确性，而我们往往会叹息道“压根儿就没有干净的数据集”。

本书正是为那些时常需要与数据打交道的人准备的，包括数据科学家、数据新闻记者、软件开发人员以及其他相关人士。无论你从事的是哪种职业，本书都会传授你一套快速而简便的实用策略，用来填补现有数据和期望数据之间的空白。人人都期盼能够拥有高质量的完美数据，但现实中的数据往往与我们的期盼相去甚远。我们是否正在饱受各种折磨呢？数据不是缺失就是格式不对，或者位置错误，还有各种异常情况，而这些问题导致的结果可以借用说唱歌手克里斯托弗·华莱士的一句歌词来表达，那就是“数据越多，麻烦越大”。

在本书中，我们始终把数据清洗当成数据科学过程中有着重要意义和重大价值的一步：轻松改进，不容忽视。我们的目标就是重新定义数据清洗，它不再是开始真正的工作之前所必须做的令人畏惧和乏味的工作。相反，我们会使用久经考验的过程与工具。我们会了解到，就好比在厨房中做菜，如果菜洗干净了，食物的色和味就不会差，我们自己也会倍感愉悦。如果再有良好的

刀工，肉就会鲜嫩可口，菜也会入味均匀。就像手艺高超的大厨们都有他们钟爱的厨具和烹饪手法一样，数据科学家们也想在绝佳的条件下处理最为完美的数据。

本书内容

第1章，*为什么需要清洗数据*。这一章通过说明数据清洗在数据科学过程中的重要作用，激发我们对干净数据的追求。随后用一个简单的例子演示了现实世界中的脏数据。在充分衡量多种清洗过程的优缺点之后，讲述了如何将清洗所带来的变化告诉其他人。

第2章，*基础知识——格式、类型与编码*。这一章介绍关于文件格式、压缩和数据类型的基础知识，同时也讨论了数据缺失和空数据，以及字符编码方面的问题。每一节都配有一个真实的案例。这一章之所以重要，是因为后面几章的学习都以这一章的基本概念为基础。

第3章，*数据清洗的老黄牛——电子表格和文本编辑器*。这一章描述了如何从常见的电子表格和文本编辑器中，尽可能多地发掘出数据清洗功能。我们还介绍了一些常见问题的简便处理方法，包括如何使用函数、搜索和替换、正则表达式来实现数据纠错和转换。在这一章的结尾处，我们将利用已经掌握的技能，使用上述两种工具来完成一个与大学有关的数据清洗任务。

第4章，*讲通用语言——数据转换*。这一章着重讨论了如何把数据从一种格式转换成另一种格式。这是数据清洗工作的重要任务之一，而我们身边各种各样的工具能帮助我们轻松完成该项任务。我们首先在几种常见的格式之间来回进行转换，如逗号分隔值（CSV）、JSON和SQL。为了演示如何在实际中使用这些技术，我们会完成一个实验项目。我们会从Facebook上下载好友关系网络数据，并把它们转换成不同的数据格式，以形象化地展现数据之间的关系。

第5章，*收集并清洗来自网络的数据*。这一章描述了三种专门针对HTML页面的数据清洗方法。其中介绍了如何利用三种流行工具从标记文本中提取数据，同时介绍了一些基本概念，以便理解其他方法。在这一章的例子中，我们会建立起一系列的清洗步骤，专门用于从网络论坛中抽取数据。

第6章，*清洗PDF文件中的数据*。这一章介绍了几种最为常见的数据顽症处理方法：提取Adobe的PDF文件中的数据。我们先采用一些低成本的工具来完成这项任务，然后再选择一些容易上手并且门槛较低的工具，最后采用Adobe自己的付费软件。所有实验都会一如既往地使用真实数据，这能让我们在学会解决问题的同时积累宝贵的经验。

第7章，*RDBMS清洗技术*。在这一章里，我们使用对公众开放的推文数据来演示多种适用于关系型数据库的数据清洗策略。例子中使用的数据库是MySQL，但其中的许多概念，如基于正则表达式的文本提取和异常检测，可以轻而易举地应用到其他存储系统中。

第8章，*数据分享的最佳实践*。这一章描述了多种分享清洗过的数据的方法，以便他人轻松

地使用你的数据。即使你暂时还没有分享数据的打算，这些方法和策略也会对你以后组织工作中的数据有所帮助。本章具体内容包括如何创建各种格式的理想数据包，如何在文档中对这些数据进行描述，如何为数据选择适合的许可协议，以及如何按需进行数据发布。

第9章，Stack Overflow项目。在这一章中我们将使用真实的数据来指导你完成一个完整的项目。在项目开始前，我们会提出一些与数据集有关的实实在在的问题。在回答这些问题期间，我们将完成第1章中所介绍的整个数据科学过程，并把在前面几章学过的清洗方法应用到其中。除此之外，为了应对庞大的数据量，我们还将采用一些新的技术来创建测试数据集。

第10章，Twitter项目。这一章描述的也是一个完整的项目，其目的是为了演示如何执行最热和变化最快的数据收集和清洗任务：Twitter挖掘。在演示中，我们将会查找并收集与某一时事相关的可公开获取的推文归档数据，同时遵守Twitter服务条款。数据采用的是目前网络API所支持的最为流行的JSON格式。在清洗和提取数据的同时，我们还将解答一个与数据集有关的简单问题。最后，我们将设计一个简单的数据模型，用它来长期存放已经抽取出来并且经过清洗的数据，并做一些简单的可视化实现。

你需要准备些什么

为了完成本书中的项目，你会用到下面这些工具和资源。

- 一款浏览器，互联网接入，现代的操作系统。我们对浏览器和操作系统本身没有什么特别的要求，但最好能支持命令行终端窗口（比如OS X上的Terminal应用）。另外，在第5章里涉及的三项活动中，有一项要依靠Chrome浏览器所提供的一个工具，所以如果你想顺利完成这项活动的话，请务必准备充分。
- 文本编辑器，如Mac OS X上的Text Wrangler，或是Windows上的Notepad++。有的集成开发环境（IDE，如Eclipse）也可以用来充当文本编辑器，但这些应用往往捆绑了太多你根本不需要的特性。
- 电子表格应用程序，如微软的Excel，或者是Google的Spreadsheets。本书中提供的例子会尽量保证在这两种工具下都可以正常运行，但有时也可能只能在一种工具下运行。
- 安装Python开发环境与Python库。我推荐使用Enthought Canopy Python环境，其地址为https://www.enthought.com/products/canopy/。
- 一个能够运行的MySQL，版本需要在5.5以上。
- Web服务器和PHP，PHP版本要求5以上。
- MySQL客户端接口，可以是命令行终端，也可以是MySQL Workbench，或者是phpMyAdmin（前提是你已经装好了PHP）。

本书的目标读者

如果你现在正在读这本书的话，我猜想你可能属于下面两类人之一。第一类是在数据清洗工作上花费了大量时间的数据科学家，希望可以进一步提高工作效率。在面对乏味单调的数据清洗任务时，你也许会觉得有些苦闷，正在寻求能够加快处理速度、提高效率的方法，或者想着是否有些别的什么工具可以立马把工作做完。在厨房的比喻中，你恰好就是那个需要提高刀工本领的大厨。

另一类是从事数据科学工作，但之前从未真正在乎过数据清洗这件事的人。但是现在，你开始琢磨了，如果事先引入清洗过程，最终得出的数据结果也许会有所改善。也许“错进，错出”这句老话让你觉得越发地真实。也许你还是一个乐于与他人分享数据成果的人，但又常常对产出的数据质量缺乏信心。通过本书，你可以学会更多的技巧并养成保持整洁的数据科学环境的习惯，随后自然可以信心满满地“当众献艺”。

但无论你属于哪一类人，本书都将帮助你重塑数据清洗的观念，让数据清洗不再是一件苦差事，而是高质量、有品位、时尚和高效的标志。你只需稍有一些编程背景即可，不要求在这方面特别强，因为在大部分数据科学项目中，发自内心的学习意愿和实验意愿，以及好奇心和对细节的注重，更为重要，也更令人推崇。

本书排版约定

本书中会出现许多用来区分不同类型信息的文本格式。接下来看一下这些格式以及它们所对应的含义。

正文中的代码、数据库表名、用户输入会以等宽字体进行表示，如：“问题是函数`open()`不能处理UTF-8编码的字符。”

代码块的表现形式如下：

```
for tweet in stream:
    encoded_tweet = tweet['text'].encode('ascii','ignore')
    print counter, "-", encoded_tweet[0:10]
    f.write(encoded_tweet)
```

另外，当我们希望你特别注意代码块中的某些部分时，相关的行或者文字会被加粗：

```
First name,birth date,favorite color,salary
"Sally","1971-09-16","light blue",129000
"Manu","1984-11-03","",159960
"Martin","1978-12-10","",76888
```

命令行中的输入和输出内容表示如下：

```
tar cvf fileArchive.tar reallyBigFile.csv anotherBigFile.csv
gzip fileArchive.tar
```

新术语和重点词汇均采用楷体字表示。

> 这个图标表示警告或需要特别注意的内容。

> 这个图标表示提示或者技巧。

读者反馈

我们总是欢迎读者的反馈。如果你对本书有些想法，有什么喜欢或是不喜欢的，请反馈给我们。这将有助于我们开发出能够充分满足读者需求的图书。

一般的反馈，请发送电子邮件至feedback@packtpub.com，并在邮件主题中包含书名。

如果你在某个领域有专长，并有意编写一本书或是贡献一份力量，请参考我们的作者指南，地址为http://www.packtpub.com/authors。

客户支持

你现在已经是Packt引以为傲的读者了，为了能让你的购买物有所值，我们还为你准备了以下内容。

彩色图片下载

我们为你准备了一个PDF文件，其中包含了本书用到的屏幕截图和图表的彩色图片。这些彩色图片将有助于你更好地理解书中的内容。该文件的下载地址是https://www.packtpub.com/sites/default/files/downloads/Clean_Data_Graphics.zip。

勘误表

虽然我们已竭尽全力确保本书内容的准确性，但仍可能有所疏漏。如果你发现了书中哪些地方有误——这些问题可能出现在正文或是代码之中——请告知我们，对此我们将万分感谢。你的

这种贡献将让其他读者免受其害，并对本书后续版本的改进有着莫大的帮助。如果你发现了错误，请通过http://www.packtpub.com/submit-errata进行提交，先选择书名，然后点击链接Errata Submission Form，就可以输入详细内容了。勘误一经核实，我们就会把它上传到我们的网站或是添加到现有勘误表中。

如果你需要查看之前已经提交的勘误信息，请访问https://www.packtpub.com/books/content/support，在搜索栏中输入书名，就可以查看现有的勘误表。

关于盗版

受版权保护的材料在互联网上遭到盗版是所有媒体都一直在面对的问题。Packt非常重视保护版权和许可证。如果你发现我们的作品在互联网上被非法复制，不管以什么形式，请立即为我们提供其网络地址或是网站名称，我们将采取相应的应对措施。

请将疑似的盗版材料链接发送至copyright@packtpub.com。

非常感谢你帮助我们保护作者，以及保护我们给你带来有价值内容的能力。

问题反馈

如果你有关于本书任何方面的问题，请联系questions@packpub.com，我们将尽快帮你解决。

目　　录

第 1 章

为什么需要清洗数据

大数据、数据挖掘、机器学习和可视化，近来计算界的几件大事好像总也绕不开数据这个主角。从统计学家到软件开发人员，再到图形设计师，一下子所有人都对数据科学产生了兴趣。便宜的硬件、可靠的处理工具和可视化工具，以及海量的免费数据，这些资源的汇集使得我们能够比以往任何一个时期更加精准地、轻松地发现趋势、预测未来。

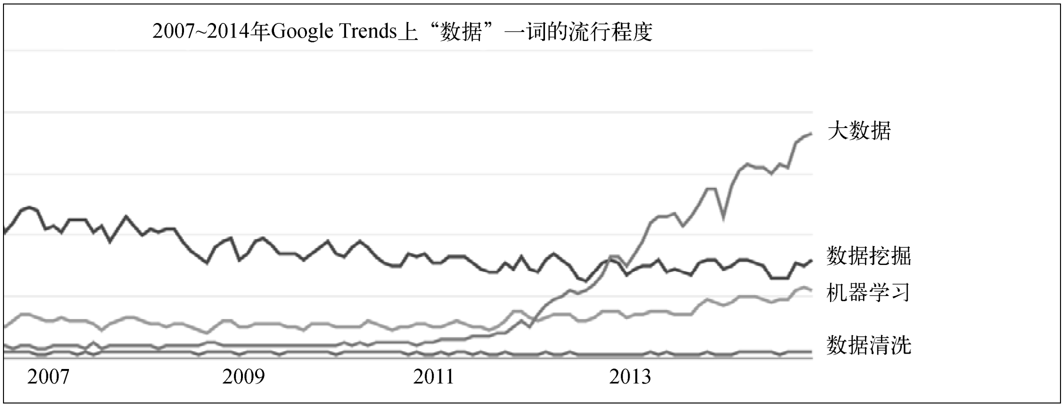
不过，你可能还未听说过的是，数据科学的这些希望与梦想都建立在乱七八糟的数据之上。在正式应用于我们认为是数据科学的核心的算法和可视化之前，这些数据往往需要经过迁移、压缩、清洗、打散、分片、分块以及其他多种转换处理。

本章内容将涵盖以下几个方面：

- 关于数据科学的六个简单处理步骤，包含数据清洗
- 与数据清洗有关的参考建议
- 对数据清洗有帮助的工具
- 一个关于如何将数据清洗融入整个数据科学过程的入门示例

1.1 新视角

最近我们读报时发现《纽约时报》将数据清洗称为看门人工作，并称数据科学家百分之八十的时间都花费在了这些清洗任务上。从下图中我们可以看出，尽管数据清洗是很重要的工作，但它并没有像大数据、数据挖掘或是机器学习那样真正地引起公众的注意。



不会真的有人因为没有见过人们聚众讨论看门人的工作多么有趣、多么酷而开始评头论足吧？说起来还真是惭愧，这工作没比做家务强到哪里去，但话又说回来，与其对它弃之不理、抱怨不断、恶语相加，还不如先把活儿干完，这能让我们过得更好些。

还不相信是吗？那让我们打个比方，你不是数据看门人，而是数据大厨。现在有人交给你一个购物篮，里面装满了你从未见过的各种各样的漂亮蔬菜，每一样都产自有机农场，并在最新鲜的时候经过人工精挑细选出来。多汁的西红柿，生脆的莴苣，油亮的胡椒。你一定激动地想马上开启烹饪之旅，可再看看周围，厨房里肮脏不堪，锅碗瓢盆上尽是油污，还沾着大块叫不出名的东西。至于厨具，只有一把锈迹斑斑的切刀和一块湿抹布。水槽也是破破烂烂的。而恰恰就在此时，你发现从看似鲜美的莴苣下面爬出了一只甲虫。

即使是实习厨师也不可能在这样的地方烹饪。往轻了说，无外乎是暴殄天物，浪费了一篮子精美的食材。如果严重点儿讲，这会使人致病。再说了，在这种地方烹饪根本毫无乐趣可言，也许全天的时间都得浪费在用生锈的破刀切菜上面。

与厨房的道理一样，事先花费些时间清洗和准备好数据科学工作区、工具和原始数据，都是值得的。“错进，错出。”这句源于上20世纪60年代的计算机编程箴言，对如今的数据科学来说亦为真理。

1.2　数据科学过程

数据清洗是如何融入数据科学中的呢？简短的回答就是，清洗工作是关键的一步，它直接影响在它之前和之后的处理工作。

稍微长一些的回答就得围绕数据科学过程的六个步骤来描述了，请看下面的列表。数据清洗正好处于中间的位置，第三步。但是，请不要以纯线性方式看待这些步骤，简单地认为这是一个从头到尾执行的框架，其实在项目的迭代过程中，我们会根据具体情况，反复执行这些步骤。另

外还需要指出的是，并不是每一个项目都会包含列表中所有的步骤。举个例子，有时候我们并不需要数据收集或可视化步骤。这完全取决于项目的实际情况。

(1) 第一步是问题陈述。识别出你要解决的问题是什么。

(2) 接下来要做的是数据收集与存储。数据从何而来？它们在哪里存放？格式又是什么？

(3) 然后是数据清洗。数据需要修改吗？有什么需要删除的吗？数据应该怎么调整才能适用于接下来的分析和挖掘？

(4) 数据分析和机器学习。数据需要哪些处理？需要什么样的转换？使用什么样的算法？运用什么公式？使用什么机器学习算法？顺序又是怎样的呢？

(5) 数据展现和可视化实现。数据处理结果应该怎样呈现出来呢？我们可以用一张或几张数据表来表现，也可以使用图画、图形、图表、网络图、文字云、地图等形式。但这是最佳的可视化方案吗？有没有更好的替代方案呢？

(6) 最后一步是问题决议。你在第一步里所提出的疑问或是问题的答案究竟是什么？数据处理结果还有哪些不足？这个方法能彻底解决问题吗？你还能找出别的什么办法吗？接下来要做的又是什么？

在数据分析、挖掘、机器学习或是可视化实现之前，做好相关的数据清洗工作意义重大。不过，请牢记，这是一个迭代的过程，因为在项目中我们可能需要不止一次地执行这些清洗操作。此外，我们所采用的挖掘或分析方法会影响清洗方式的选取。我们可以认为清洗工作包含了分析方法所能决定的各种任务，这有可能是交换文件的格式、字符编码的修改、数据提取的细节等。

数据清洗与数据收集和存储（第2步）的关系也十分密切。这意味着你得收集原始数据，对它们执行存储和清洗操作，之后再把清洗过的数据保存下来，接下来收集更多的数据，清洗新的数据并把清洗结果与前面处理完的结果数据结合起来，重新进行清洗、保存等操作，反反复复。正因为这个过程非常复杂，所以我们要么选择牢牢记住曾经做过的处理，并记录下那些可以根据需要反复执行的步骤，要么把工作的全部状况告知其他相关人员。

1.3　传达数据清洗工作的内容

六步处理过程是围绕着问题和解决方案这个故事线组织的，因此，在作为报表框架使用时，它的表现十分优秀。如果你已经决定使用六步框架来实现数据科学过程报表，将发现只有到了第三步你才会真正开始进行与清洗有关的工作。

哪怕你并不需要把数据科学过程制成正式的报告文档，你仍然会发现，认真地记录下曾经按什么顺序做了些什么事情，对以后的工作也是极有帮助的。

请记住，哪怕是规模再小、风险再低的项目，你也要面对至少两人规模的受众：现在的你和六个月之后的你。请相信我说的话，因为六个月之后的你基本上不会记得今天的你做过什么样的清洗工作，也不记得其中的缘由，更谈不上如何重新再做一次。

要解决这个问题，最简单的方案就是保留一份*工作日志*。这个日志应该包含链接、屏幕截图，或是复制粘贴你曾经运行过的具体的命令，并配上为什么要这样做的解释性文字。下面是一个关于小型文本挖掘项目的日志示例，其中记述了每个阶段输出的外部文件链接以及相关的清洗脚本链接。如果你对日志中提到的某些技术不太熟悉的话，也没有关系，因为这个示例的重点只是让你了解一下日志的样子而已。

(1) 我们写了一条SQL查询语句来检索出每条数据及其相关描述。

(2) 为了能在Python中进行词频分析，我们需要把数据调整成JSON格式。因此我们做了一个PHP脚本，用它来循环遍历查询结果，并以JSON格式保存到文件中（第一个版本的数据文件）。

(3) 这个文件里的数据有些格式上的错误，比如包含了没有转义的问号和一些多余的内嵌HTML标签。这些错误可以在第二个PHP脚本中修正。运行第二个脚本之后，我们就可以得到一份干净的JSON文件了（第二个版本的数据文件）。

这里需要注意的是，我们用日志来解释程序做过什么和这样做的原因。日志的内容可以很简短，但要尽可能地包含一些有用的链接。

另外，我们还可以选择许多更复杂的方案来传达信息。例如，如果你对软件项目管理中常用的*版本控制系统*比较熟悉的话，如Git或是Subversion，就可以好好地规划设计一番，想想怎么使用这些工具来跟踪数据清洗工作。不管你使用什么样的系统，最重要的事情是做好日志，哪怕只有一句话。来吧，学着把它用起来，别耽误进度了。

1.4　数据清洗环境

本书中涉及的数据清洗方法是通用的，适用范围非常广泛。你不需要任何高端专业的数据库产品或是数据分析产品（事实上，这些厂商和产品可能已经提供了数据清洗程序或是解决方法）。我围绕数据处理过程中的常见问题，设计了本书中的清洗教程。而我要展示的都是适用范围较为广泛的开源软件和技术，它们很容易在实际工作中获得和掌握。

下面列出了你需要准备的工具和技术。

- 几乎在每一章中，我们都会用到终端窗口和命令行界面，比如Mac OS X上的Terminal程序或者是Linux系统上的bash程序。而在Windows上，有些命令可以通过Windows的命令提示符运行，但其他的命令则需要通过功能更强的命令行程序来运行，比如CygWin。
- 几乎在每一章中，我们还会用到文本编辑器或者是适合程序员使用的编辑器，如Mac上的

Text Wrangler，Linux上的vi或emacs，或是Windows上的Notepad++、Sublime编辑器等。

- 在绝大数章节里，我们需要使用Python 2.7版本的客户端程序，如Enthought Canopy，另外还需要足够的权限来安装一些包文件。其中大部分例子都可以直接在Python 3中运行，但有些不可以，所以如果你安装的是Python 3的话，可以考虑再安装一个2.7版本。
- 在第3章“数据清洗的老黄牛——电子表格和文本编辑器”中，我们需要使用电子表格程序（主要是Microsoft Excel和Google Spreadsheets）。
- 在第7章“RDBMS清洗技术”中，我们需要使用MySQL数据库和一个用于访问该数据库的客户端软件。

1.5 入门示例

准备好了吗？现在让我们磨好手里的厨刀并结合六步框架来解决一些简单的数据清洗问题吧。这个例子会用到对公众开放的安然（Enron）公司电子邮件数据集。这是一个非常有名的数据集，当中所有的往来邮件都源自现已停业的安然公司前雇员。作为美国政府调查安然公司账目欺诈的一部分，雇员之间的邮件已被公开并可供任何人下载。来自各个领域的研究人员已经发现，这些邮件有助于研究商务沟通、社交网络等问题。

> 你可以在维基百科http://en.wikipedia.org/wiki/Enron上阅读更多关于安然公司和导致它破产的金融丑闻。在另外一个页面http://en.wikipedia.org/wiki/Enron_Corpus上，你可以阅读关于安然公司电子邮件语料库的信息。

在这个例子当中，我们会采用六步框架来解决一个简单的数据科学问题。假设我们需要揭露在某一段时间里安然公司内部电子邮件的使用趋势和特征。先让我们按照日期来对安然雇员之间的往来邮件数量做个统计，然后再通过图形来显示统计出来的数据。

首先，我们需要按照http://www.ahschulz.de/enron-email-data/上面的指南下载一份MySQL版本的安然公司语料库。另一个（备份）源的地址是https://www.cs.purdue.edu/homes/jpfeiff/enron.html。根据指南，我们需要把数据导入到MySQL服务器中一个称为Enron的数据库模式中。导入之后的数据可以通过MySQL命令行界面或是基于网页的PHPMyAdmin工具进行查询。

这是我们的第一个数据统计查询，语句如下：

```
SELECT date(date) AS dateSent, count(mid) AS numMsg
FROM message
GROUP BY dateSent
ORDER BY dateSent;
```

从结果中我们马上就会注意到，许多邮件的日期都不正确。比如，很多日期要么早于或晚于公司的存续期（如1979），要么与事实逻辑不符（如0001或2044）。邮件虽旧，但也不至于那么旧！

下表是截取出来的一部分数据片段（完整的结果集长达约1300行）。这些数据的日期格式都是正确的。但是，有些日期值有着明显的错误。

dateSent	numMsg
0002-03-05	1
0002-03-07	3
0002-03-08	2
0002-03-12	1
1979-12-31	6
1997-01-01	1
1998-01-04	1
1998-01-05	1
1998-10-30	3

这些错误日期的产生很有可能是由邮件客户端配置不当导致的。这里，我们有三种处理方案可以选择。

- 什么都不处理：也许，我们可以选择忽略这些错误数据，直接开始构建线性图。但是，最小的错误日期始于0001年，最大的错误日期至2044年结束，所以我们可以想象，最终的线性图时间轴上将有1300个刻度线，每个刻度上面显示着1或者2。光是听起来就没有什么吸引人的地方，也没提供什么有用的信息，所以不处理就等同于数据毫无用处。
- 修正数据：我们可以尝试算出错误消息对应的正确日期，从而生成正确的数据集来创建图形。
- 扔掉受影响的邮件：我们可以做出一个明智的决定，放弃那些日期不在预定范围之内的邮件。

为了在选项二和选项三之间做个决断，我们需要先计算一下1999~2002年有多少封邮件会受到影响。为此，我们编写了下面的SQL语句：

```
SELECT count(*) FROM message
WHERE year(date) < 1998 or year(date) > 2002;
Result: 325
```

结果显示一共有325封邮件包含日期错误，乍一看确实有点多，但请等一下，实际上这些数据只是占了全部数据量的百分之一。现在需要好好斟酌一下了，或许我们可以手工修复这些日期，但也可以假定不在乎这百分之一的损失。那就干脆扔掉这些数据直接选择第三个方案吧。下面是调整后的查询语句：

```
SELECT date(date) AS dateSent, count(mid) AS numMsg
FROM message
WHERE year(date) BETWEEN 1998 AND 2002
GROUP BY dateSent
ORDER BY dateSent;
```

数据清洗完成之后一共生成了1211条结果，每一行都有其对应的邮件数量。下面的内容截取自新的数据结果集：

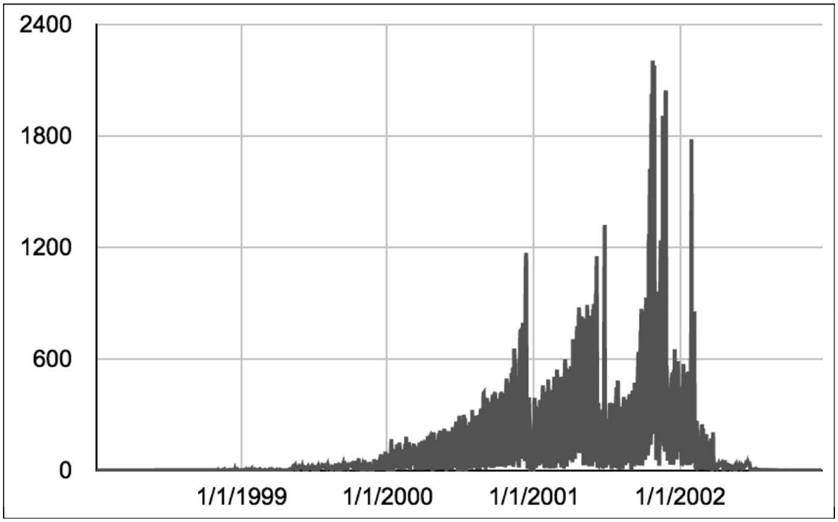
dateSent	numMsg
1998-01-04	1
1998-01-05	1
1998-10-30	3
1998-11-02	1
1998-11-03	1
1998-11-04	4
1998-11-05	1
1998-11-13	2

在新的数据中，1998年1月的两个日期看起来有些问题，因为其他邮件都始于10月，而且10月之后的邮件数量更为规律一些。这很奇怪，同时也反映了另一个问题：我们是否还有必要在*x*轴上标记每一个日期，即使这一天一封邮件都没有发出？

如果我们的答案是肯定的话，那就需要显示每一个日期，即使它对应的邮件数量是0。这意味着我们需要再做一轮数据清洗工作，生成那些没有邮件往来的日期所对应的数据。

但是请等一下，对于这个问题的处理，我们可以稍微变通一下。是不是真的要在原始数据中加入零数据，其实这取决于我们用什么样的工具来创建图表以及图表的种类。举个例子来说，Google的Spreadsheets就能在初始数据中缺少日期的情况下，在*x*轴上自动进行零值数据补齐，创建线性图或是条状图。在我们的数据中，这些需要补齐的零值就是1998年所缺失的日期。

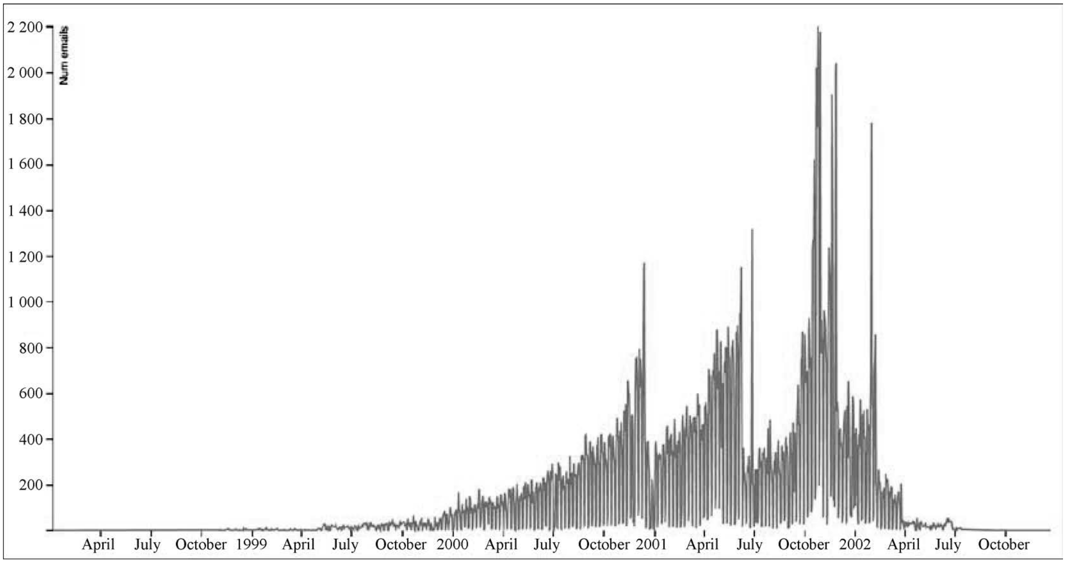
下面的三幅图演示了这些工具所呈现的结果，并反映出它们是如何处理日期轴上的零值数据的。请注意，Google Spreadsheets在头部和尾部所展现的长长的零值数据。



Google Spreadsheets自动为缺失的日期补齐零值数据

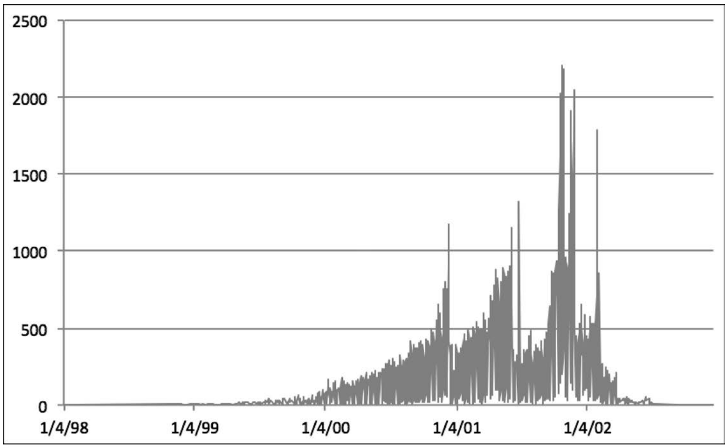
D3 JavaScript可视库也能完成同样的工作，零值数据也是自动补齐的，如下图所示。

关于如何使用D3完成一个简单的线性图，请参考教程：http://bl.ocks.org/mbostock/3883245。



D3自动为缺失的日期补齐零值数据

Excel的线性图也有同样的数据补齐功能，如下所示。

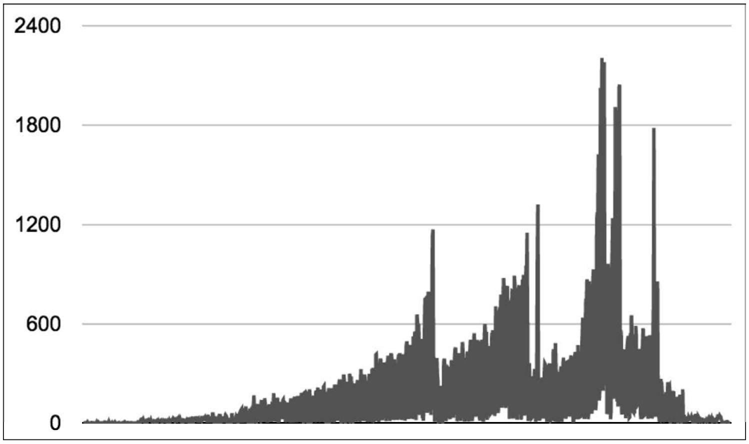


Excel自动为缺失的日期补齐零值数据

接下来，我们需要考虑一下是否需要零值数据，要不要让它们在x轴上显示（数据查询结果总数为1211，而在指定的年限范围，即1998~2002年，一共有1822天）。或许显示零值日期并不会

起到什么作用；此外，倘若图表过于拥挤，有些较小的空隙我们就看不到了。

为了比较两种图表之间的差别，我们可以快速地把相同的数据再次放入Google Spreadsheets（你也可以在Excel或D3中完成这个任务），但这次我们只需选择数量这一个字段来构建图表就可以了，这样一来，Google Spreadsheets就不会在x轴上显示日期了。最后生成的结果图表中只包含数量数据，零值数据不会被填充进来。长长的尾巴不见了，图表中各个重要的部分（中间部分）也都被保留了下来。



图表现在只显示有一条或多条邮件消息的记录

值得庆幸的是，这次生成的图表与之前的相比还是很像的，并省去了前后两端冗长的内容。根据比较结果和我们先前的计划（还记得我们要做的只是创建一个简单的条形图吧），现在终于可以继续下一步工作，不用烦恼是否需要创建零值数据了。

当一切都完成之后，最终的条形图会为我们显示出安然公司曾经的几次邮件峰值。最大的峰值出现在2001年10月和11月，恰好是丑闻被揭露的时候。还有两次较小一点的峰值发生在2001年6月26日和27日，以及2000年12月12日和13日，那时的安然也有新闻事件发生（分别是加利福尼亚州的能源危机事件和公司领导层的变动）。

如果数据分析已经让你开始兴奋的话，那么接下来你可以好好利用这些数据来发挥你的奇思妙想。愿这些清洗干净的数据可以让你的分析工作更上一层楼！

1.6　小结

当所有工作全都做完的时候，事实印证了《纽约时报》的报道。我们从这个简单的练习中可以看出，即便是回答一个小小的数据问题，数据清洗就占了整个过程80%的工作量（在这个全文

共计900个单词的案例中，光是谈论数据清洗的基本原理和方案就用了700个单词）[①]。数据清洗的的确确是数据科学过程的关键部分，它不仅涉及对技术问题的理解，同时还要求我们做出相应的价值判断。作为数据清洗工作的一部分，我们甚至需要在尚未完成分析与可视化步骤的时候，预先考虑它们的输出结果将是什么样子。

重新审视数据清洗在这一章的工作中所扮演的角色，我们很容易发现，清洗效果的提升能够大幅减少后续处理的时间。

在接下来的一章里，我们将学习一些与数据有关的基础知识，这是每一个“数据主厨”在进入既宽敞又明亮的大“厨房”之前需要掌握的，内容包括文件格式、数据类型和字符编码。

① 这里指的是英文原版字数统计。——译者注

第 2 章

基础知识——格式、类型与编码

几年前，在一次家庭的节日聚会上，我收到了一件非常有趣的礼物。那是一个冷餐厨房专用的工具箱，里面有各种不同用途的工具，切刀、去皮器、勺子、果皮刮刀等一应俱全。使用一段时间之后，我慢慢地学会了每一种工具的使用方法，并自创了一套刨丝刀和番茄刀的使用窍门。所以，我在这一章也为你准备了一套适用于数据清洗的入门工具。我们要学习的内容有：

- 文件格式，包括压缩标准
- 数据类型的基础知识（包括多种用于表示缺失数据的类型）
- 字符编码

这些基础知识都是我们需要掌握的，因为后续章节里的内容都建立在这些基础知识之上。其中有一些特别基础的概念，你几乎天天都能碰上，比如文件压缩和文件格式。它们就像大厨们用的普通刀具那样司空见惯。不过还是有一些专业性和目的性都比较强的概念，比如字符编码，丝毫不逊色于番茄刀！

2.1 文件格式

在这一节中，我们将描述数据科学家们在处理实际数据时常常会遇到的五花八门的文件格式，换句话来讲，这些数据在许多书的示例数据集中都很难找到，因为书里的数据格式都是经过精心整理而编排出来的。在这里，我特意设计了一些场景来帮助你理解每一种文件格式的应对策略以及它们的局限性，并在之后讨论多种不同的文件压缩格式和文件归档格式。

2.1.1 文本文件与二进制文件

从网络上收集数据的时候，你可能遇到过下面几种情况。

- 数据可以以文件形式进行下载。
- 数据可以通过存储系统的交互式界面进行访问，比如利用查询接口来访问数据库系统。

- 数据可以通过持续不断的流的形式进行访问。
- 数据可以通过应用编程接口（API）来访问。

但无论在哪种情况下，要想把这些数据分享给其他人，我们都需要亲自动手创建自己的数据文件。因此，牢牢地掌握好各种数据文件格式并熟知它们的优缺点是至关重要的。

首先，让我们考虑一下计算机系统中广义存在的两种文件类型，也就是人们常常提起的文本文件和二进制文件。由于所有的文件都是由一系列字节一个挨着一个紧密排列组合而成的，所以，严格地说，所有文件都是二进制的。但是，如果文件中的字节都以纯粹的字符形式保存（例如字母、数字，或是换行、回车、制表符这样的控制字符），那么我们就可以说这个文件是文本格式的。相比之下，二进制文件包含的字节则是由大部分非人类可读的字符组成的。

1. 文件的打开与读取

文本文件可以通过一种称为文本编辑器的程序读取和写入。如果你尝试着在文本编辑器中打开一个文件，并成功读取了里面的内容（即便你不太理解里面的内容），那么这个文件很可能就是一个文本文件。但是，如果你在文本编辑器打开一个文件，内容看起来全是乱七八糟的字符和奇怪的非法字符，那么它可能就是一个二进制文件。

二进制文件只能通过特殊的应用程序打开或编辑，而不是文本编辑器。例如，Microsoft Excel文件需要使用Microsoft Excel电子表格程序才能打开和读取，数码相机拍摄的照片文件需要使用图形程序来读取，如Photoshop或Preview等。有些时候，二进制文件也可以被多个兼容的软件包读取，比如许多不同的图形程序都能读取和编辑照片文件。另外还有一些二进制文件，它们虽然是某些程序的专属格式，但也可以通过别的兼容软件打开，以Microsoft Excel或Microsoft Word文件为例，它们就可以通过Apache的OpenOffice打开。除此以外，还有一些二进制编辑器，通过它们你可以探访二进制文件的内部并对其进行深度编辑。

在某些情况下，文本文件既可以供应用程序读取，同时也可以由文本编辑器来读取。举例来说，网页和计算机的程序源代码本身都是纯文本字符，我们很容易在文本编辑器里编辑它们，不过这要求你已经掌握了格式、布局和语义相关的知识，否则其中的内容是很难理解的。

其实大多数时候，想要知道一个文件的类型是不需要打开文件的。举例来说，人们在查找文件的时候一般都是从文件名开始。由三个字母和四个字母组成的文件扩展名就是一种表明文件类型的常见方式。我们经常见到的文件扩展名有：

- 以.xlsx为扩展名的Excel文件、以.docx为扩展名的Word文件、以.pptx为扩展名的Powerpoint文件；
- 以.png、.jpg和.gif为扩展名的图形文件；
- 以.mp3、.ogg、.wmv和.mp4为扩展名的音乐和视频文件；
- 以.txt为扩展名的文本文件。

在互联网上，有的网站列出了常见文件扩展名以及与之有关的应用程序，其中比较有名的是

fileinfo.com。另外在维基百科上还有文件扩展名的字母列表，其访问地址为https://en.wikipedia.org/wiki/List_of_filename_extensions_(alphabetical)。

2. 深入文件内部

如果你迫切希望打开一个未知的文件，对其内容一探究竟的话，那么我在这里准备了几个命令行工具供你使用，利用它们你可以看到文件开头的一些字节。

(1) 在OS X或Linux中

在Mac OS X或者是Linux系统上，打开终端窗口（在Mac上，你可以通过Applications | Utilitiies | Terminal找到标准的终端应用程序），使用工作目录打印命令（`pwd`）和目录切换命令（`cd`）进入文件的所在位置，然后使用`less`命令逐页查看文件的内容。下面是我完成这个任务所使用过的命令：

```
flossmole2:~ megan$ pwd
/Users/megan
flossmole2:~ megan$ cd Downloads
flossmole2:Downloads megan$ less OlympicAthletes_0.xlsx
"OlympicAthletes_0.xlsx" may be a binary file. See it anyway?
```

如果程序提示你“view anyway”的话，那么这个文件就是一个二进制文件，你应该做好看到杂乱字符的心理准备。你可以通过键盘输入字母`y`继续查看该文件，也可以输入字母`n`拒绝继续查看。下图是用`less`命令查看文件OlympicAthletes_0.xlsx的结果。结果果然是一团糟吧！

```
PK^C^D^T^@^F^@^H^@^@^@!^@@J)<CF>r^A^@^@^P^E^@^@^S^@<D9>^A[Content_Types].xml
<A2><D5>^A(<A0>^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<AC><94>AN
<C3>0^PE<F7>H<DC>!<F2>^V%nY <84><9A>vQ`      <95>(^G0<F6><A4><B1><EA><U+0616><C7>
-<ED><ED><99>$4<A2><A8>^DU<E9>&Q^T<CD><FF>^?99>=G<ED>*<93>l!<A0>v6g<E3>l<C4>^R
<B0><D2>)mW9{_><A7><F7>,<C1>(<AC>^R<C6>Y<C8><D9>^^<90>-<D7   <93><E5><DE>^ &Tm1ge
<8C><FE><81>s<94>%T^B3<E7><C1>k<U+0085>JD<FA>^L+<EE><85>\<8B>^U<F0><DB><D1><E8>
<8E>Kg#<U+0618><C6>Z<83>M'<AF>^T h^E<C9>B<84><F8>"*<F2><E1>;<C3>#<A9>A<FB>^\g
<A4>ǒy[X{<E7>Lxo<B4>^T<91><92><F3><AD>U<BF>\SW^TZ<82>rrS<91>W<U+0588><DD><D4>*
<FC>OC<8C>{^C8<D8>
}^@<A1><B0>^D<88><95><C9>Zy<F3>#^Tbcb<F2><B4>#^B-<F4>^@^F<CF>k<ED>ESCfF<95>M<FB>
Xj<8F>=^N<FD><EC><FA><99>|<BA><B0><FE>pn}i*5<9D><AC>^R<DA>^^r<9F>Z^B<9A><DE>"8
<8F><9C>f=8^@<D4><C8>^U<A8>Л$<84><A8><A1>cvG^V<B0><EE><BD>^Y#<F2><E6>5|^K<8F>W
:
```

查看完文件内容之后，你可以输入字母q退出less程序。

(2) 在Windows中

在Windows的命令提示符里有一个对应的more程序。操作方法与之前介绍过的less命令类似（这正应了那句俗语——less is more）。你可以从Windows 7的开始菜单中运行cmd命令打开命令提示符程序。而在Windows 8中，则需要通过“应用程序 | Windows系统 | 命令提示符”。当然，我们还是要像前一个例子那样使用cd和pwd找到目标文件。

2.1.2　常见的文本文件格式

在本书中，我们最为关心的是文本文件，而非二进制文件。（但压缩文件和归档文件这样的特殊情况除外，这些文件会在下一节讨论，至于Excel文件和PDF文件会在本书后面的章节里分别进行讨论。）

本书所关注的文本文件类型主要有三种：

- 分隔格式（结构化数据）
- JSON格式（半结构化数据）
- HTML格式（非结构化数据）

这些文件有着各自不同的布局格式（主要是在文件读取的时候内容的表现形式不同），结构化程度也有差异。换言之，就是每种文件的内容有多大比例是以结构化形式进行数据组织的？又有多少数据是无规则的或无结构的？在此，我们会分别讨论每种文件格式。

2.1.3　分隔格式

分隔文件在数据分享和传输中使用得非常广泛。分隔文件就是文本格式文件，数据属性（列）和数据实例（行）由统一的符号分隔。我们把分隔用的字符称为*分隔符*。最为常见的分隔符是制表符和逗号。这两种方案分别出现在*制表符分隔值*（TSV）和*逗号分隔值*（CSV）中。有时候，分隔文件也被称为*记录式文件*，因为文件中的每一行都代表了一条记录。

下面的示例中列举了三条记录（这三行记录描述了三个人的信息），每条记录的值都采用逗号分隔。第一行列出了各个字段的名字。第一行也被称为*标题行*，并高亮显示，请见下面的内容：

```
First name,birth date,favorite color
Sally,1970-09-09,blue
Manu,1984-11-03,red
Martin,1978-12-10,yellow
```

请注意例子中的分隔数据，它们不包含任何*非数据*信息。其中的内容要么代表完整的一条行数据，要么代表着独立的字段数据。这些数据的结构化程度非常高。但这并不妨碍我们从分隔格式中

辨识出不同的数据。最为明显的区分点就是每个数据实例（每一行）彼此之间都是分离的。通常在一行的末端会出现换行符或回车符，或者是两者同时出现，这取决于文件创建时的操作系统环境。

1. 查看不可见字符

在前面的例子中，换行符和回车符都是看不见的。怎么才能看到不可见字符呢？我们将在Mac系统中使用Text Wrangler（这类似于Windows上的Notepad++）重新读取例子中的文件，并通过Show invisibles的功能（可以通过View | Text Display激活）来查看不可见字符。

```
First·name,birth·date,favorite·color¬
Sally,1970-09-09,blue¬
Manu,1984-11-03,red¬
Martin,1978-12-10,yellow¬
```

此外，还有一种可以查看不可见字符的方法，那就是在Linux系统或Mac上的Terminal窗口中使用vi（一种命令行文本编辑器）工具。使用vi查看文件里不可见字符的过程如下。

(1) 首先，输入下面的命令：

```
vi <filename>
```

(2) 然后输入:进入vi的命令模式。

(3) 接下来输入set list并按回车键，这时就能看到不可见字符了。

下面的截图演示的是通过set list命令在vi里显示行尾字符，其对应符号为$。

```
First name,birth date,favorite color$
Sally,1970-09-09,blue$
Manu,1984-11-03,red$
Martin,1978-12-10,yellow$
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:set list
```

2. 封闭错误数据

在分隔文件中，除了分隔符外，另一个重要的选项是用什么字符来封闭分隔数据。举个例子来说，逗号分隔符对一般值都有效，但碰到含有逗号作为千位分隔符的数值时就束手无策了。请看下面的数据，最后一个字段工资所对应的数据就含有千位分隔符，而这个符号同时还肩负着数据分隔的作用：

```
First name,birth date,favorite color,salary
Sally,1971-09-16,light blue,129,000
Manu,1984-11-03,red,159,960
Martin,1978-12-10,yellow,76,888
```

如何在创建文件时搞定这个问题呢？其实，有两种方案。

- **方案一**：分隔文件创建者需要在整个数据表创建之前，删除最后一列中的逗号（也就是说工资中不可以包含逗号字符）。
- **方案二**：分隔文件创建者需要使用额外的符号来对数据进行封闭处理。

如果选择了第二种处理方案，通常的做法是采用双引号来对数据进行封闭。这样一来，第一行末端出现的`129,000`就会被改成`"129,000"`。

3. 字符转义

如果数据本身就含有引号，这种情况该如何应对呢？比如在下面的数据中，Sally喜欢的颜色列表中有`light "Carolina" blue`，该怎么解决呢？

```
First name,birth date,favorite color,salary
"Sally","1971-09-16","light "Carolina" blue","129,000"
"Manu","1984-11-03","red","159,960"
"Martin","1978-12-10","yellow","76,888"
```

每当这个时候，就需要使用另外一个特殊字符来对数据内部使用的引号进行转义，而转义字符就是反斜线`\`：

```
First name,birth date,favorite color,salary
"Sally","1971-09-16","light \"Carolina\" blue","129,000"
"Manu","1984-11-03","red","159,960"
"Martin","1978-12-10","yellow","76,888”
```

> 也许你会想到改用单引号来替代双引号作为封闭符号，但这种方法又有可能碰上所有格“it’s”，或是人名O’Malley这样类似的问题。总之，问题总是会存在的。

分隔文件使用起来非常方便，因为它们的格式很容易理解，访问起来也很方便，一个普通的文本编辑器就能搞定它。但是，为免碰到类似上面介绍的那些情况，就得事先做好一番规划，以

确保数据能够被正确无误地分隔。

如果在工作过程中，你碰巧发现在你接收的数据中含有分隔错误，那么你可以参考第3章中所教授的技巧来清洗这些数据。

4. JSON格式

JSON（JavaScript Object Notation），发音为JAY-sahn，是时下最为流行的数据格式之一，这种格式的数据有时也被称为半结构化数据。JSON的名字虽然包含JavaScript字样，但它并非只限于在JavaScript中使用。名字只是陈述了该类型被设计用于序列化JavaScript对象这个事实而已。

半结构化数据集的特点是数据的值都有其对应的属性标识，而且顺序无关紧要，有时甚至可以缺失某些属性。JSON就是这样以属性–值为基础的数据集，示例如下：

```
{
  "firstName": "Sally",
  "birthDate": "1971-09-16",
  "faveColor": "light\"Carolina\" blue",
  "salary":129000
}
```

属性被放在冒号的左边，值被放在冒号的右边。每个属性之间都采用逗号进行分隔。整个实体使用花括号进行封闭处理。

JSON在有些方面与分隔文件类似，但也有不同的地方。首先，字符串值必须使用双引号进行封闭处理，因此，字符串内部的双引号也都必须用反斜线进行转义。（转义字符也是\，如果这个字符被当作数据内容使用，它本身也必须被转义！）

在JSON中，逗号不可以出现在数字类型的数据中，除非这个值被当作字符串使用并用引号封闭。（在对数字进行字符化处理时一定要小心——你确定要这样做，是吧？如果你已经决定了，请参考2.3.1节，其中的内容有助于你解决相关的问题。）

```
{
  "firstName": "Sally",
  "birthDate": "1971-09-16",
  "faveColor": "light\"Carolina\" blue",
  "salary": "129,000"
}
```

JSON还支持多值属性和多层级结构（这两种功能在分隔文件中实现起来比较困难）。请看下面的JSON文件，它含有一个多值属性的`pet`字段，还有一个带有层级结构的`jobTitle`字段。这里需要注意的是，之前的`salary`字段已经被转移到新的`job`字段里了：

```
{
  "firstName": "Sally",
  "birthDate": "1971-09-16",
  "faveColor": "light\"Carolina\" blue",
  "pet":
```

```
    [
      {
        "type": "dog",
        "name": "Fido"
      },
      {
        "type": "dog",
        "name": "Lucky"
      }
    ],
    "job": {
      "jobTitle": "Data Scientist",
      "company": "Data Wizards, Inc.",
      "salary":129000
    }
  }
```

- JSON实验

JSON作为数据交换格式极为流行，这是因为它扩展性好，容易使用，并支持多值属性、可缺失属性、嵌套属性/层级属性。各种分布式数据集所采用的API也对JSON的发展有着极大的贡献。

接下来让我们用iTunes API做个实验，看看如何利用搜索关键词来生成JSON数据结果集。iTunes是由Apple公司提供的一个音乐服务。任何人都可以利用iTunes服务来查找歌曲、艺术家和专辑。在查找的时候我们需要把搜索关键词添加到iTunes API URL的后面：

https://itunes.apple.com/search?term=the+growlers

在这个URL中，=后面的内容就是搜索关键词。我搜索的是我比较喜欢的一个乐队，名为Growlers。这里需要注意的是，URL中含有一个用来代替空格字符的+，因为URL是不允许包含空格字符的。

iTunes API会根据我提供的关键词从它的音乐数据库里返回50个结果。整个结果集会被格式化成一个JSON对象。结果集中的每一个元素都是以名字–值格式存在的JSON对象。这个例子返回的JSON数据貌似很长，因为结果记录有50个之多，但实际上每个结果都特别简单——因为在这个URL中展现出来的iTunes数据没有复杂的多值数据和层级数据。

> 关于iTunes API更多的使用细节，请访问Apple iTunes的开发文档：https://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html。

5. HTML格式

HTML文件，或称为网页文件，也是一种文本格式文件，其中经常夹杂着各种冗余的数据。

其实大部分人都可能有过这样的经历，在浏览网站时发现一些内容丰富的表格或是列表，想把它们“据为己有”。有时只需通过简单地复制粘贴便可以从网页中创建我们想要的数据分隔文件，但大多数时候复制粘贴是不起作用的。因为许多HTML文件内容都混乱无比，所以抽取数据的过程也是苦不堪言。所以有时我们会说，网页文件是无结构的数据文件。虽然网页可以通过HTML标签对其数据进行整理，但实际上并不是所有的网页内容都是这样组织的。另外，很多HTML标签的使用也极其不规范，甚至是充满了各种错误，这种现象不单单体现在不同网站之间，即便是同一家网站也有类似的情况发生。

下面的截图是http://weather.com网站的一部分。尽管其中的内容包括了图片、色彩和其他非文本内容，但说到底，这个网站店内容还是使用HTML编写的，只要我们想要从页面中提取文本数据，就一定能做得到。



虽然HTML源代码接近1000行，但只要我们足够细心，完全可以从中找出让浏览器呈现出天气表格的数据和布局指令：

```
<div class="wx-temperature-label">FEELS LIKE
<span itemprop="feels-like-temperature-fahrenheit">43</span>&deg;</div>
</div>
<div class="wx-data-part">
<div class="wx-temperature">43<span class="wx-degrees">&deg;</span></div>
<div class="wx-temperature-label">HIGH AT 1:25 PM</div>
</div>
<div class="wx-data-part">
<div class="wx-temperature">28<span class="wx-degrees">&deg;</span></div>
<div class="wx-temperature-label">LOW</div>
</div>
```

看到了吧，毫无结构化可言！是的，从技术上讲，我们的确可以从这个页面抽取数据43（华氏温度），但这个过程并不容易，因为我们没法保证今天使用的方法可以适用于明天的情况，或是后天的情况，因为http://weather.com网站可以随时改变它的源代码。除此之外，页面上还混杂着许多其他数据，所以我们将会在第5章讨论如何从基于网页的非结构化文件中抽取数据。

2.2　归档与压缩

在什么情况下，一个文件既是文本文件，同时也是二进制文件呢？当然是压缩或归档的时候了。那到底什么是压缩和归档呢？在这一节中，我们就学习一下文件的压缩和归档，并了解一下它们的工作原理和各自对应的不同标准。

这一节的内容是很重要的，因为现实中的许多数据（特别是分隔数据）都是以压缩文件的形式存在的。数据科学家最常用的数据压缩格式都有哪些？在这里我们会找到想要的答案。你也许还琢磨着怎么把文件压缩完再分享给别人。该怎么评估哪种压缩方法是最佳选择呢？

2.2.1　归档文件

归档文件就是一个内部包含了许多文件的独立文件。这种文件的内部可以包含文本文件或二进制文件，或者是两者的混合。归档文件是使用一个特殊程序将给定的文件列表转换成一个文件而制成的。当然，归档文件也可以通过反向操作被转换成多个文件。

tar

在做数据科学工作时，你碰到最多的归档文件可能就是*磁带归档*（TAR）文件了。这种文件是通常是用tar程序创建的，并以.tar为后缀名。它的最初设计目的是用于磁带归档技术。

tar程序在类Unix操作系统中都存在，所以我们也可以在Mac OS X的Terminal程序中使用它。

要想创建tar文件，你需要给tar程序下达几个指令，让程序知道哪些文件是你想要包含的，输出的文件名字又是什么。（程序中的选项`c`用来表明我们想创建一个归档文件，选项`v`可以让程序在加载文件列表的时候打印出文件名字，选项`f`用于指定输出文件的名字。）

```
tar cvf fileArchive.tar reallyBigFile.csv anotherBigFile.csv
```

如果想要“untar”一个文件（也就是把归档文件展开，提取出里面的文件），你只需要让tar程序指向你想要展开的文件即可。其中`xvf`中的字母`x`表示提取（eXtract）的意思：

```
tar xvf fileArchive.tar
```

现在我们已经知道一个.tar格式的归档文件里面含有多个文件，但到底有多少文件，它们又都是什么样的文件呢？在提取文件之前，你可能想要先掌握一些基本信息，比如即将被提取的文件是我们想要的吗？是否还有足够的磁盘空间来存放提取出来的文件？这时tar命令中的选项`t`就可以派上用场了，它会以列表形式显示tar文件中的内容：

```
tar -tf fileArchive.tar
```

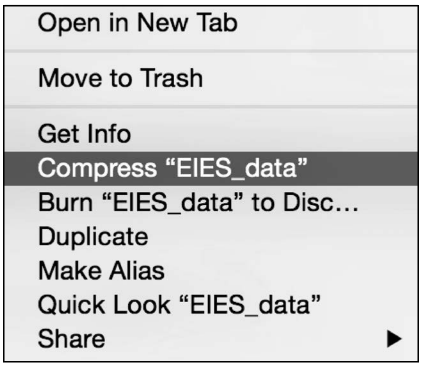
除了tar以外还有许多别的归档程序，当中包括一些具备压缩功能的程序（比如，OS X系统上内置的ZIP压缩软件和Windows上的各种ZIP和RAR工具），它们在做归档的同时还能进行文件压缩处理，接下来我们就来看看压缩的概念。

2.2.2 压缩文件

压缩文件的作用就是缩减原始文件的大小，使得它们占用较少的存储空间。文件变小意味着占用的磁盘空间更少，在网络间共享传输会更快。而在数据文件处理方面，我们关注的重点就是如何轻松地进行数据文件压缩，并把压缩文件重新还原成原始文件。

1. 如何压缩文件

创建压缩文件的方法有许多，选择哪种取决于你所使用的操作系统和你安装的压缩软件。比如在OS X上，从Finder程序中随便选中哪个文件或文件夹（或组），然后点击鼠标右键，再从菜单中选择Compress。操作完成后就会在被压缩文件所在的相同目录下出现一个压缩文件（.zip后缀名）。操作截图如下所示。



2. 如何解压文件

在数据科学过程中的数据收集步骤里，经常会处理那些下载下来的压缩文件。这些文件有可能是文本分隔文件，比如像本章前面描述的那样，也有可能是其他类型的数据文件，比如构建数据库用的电子表格或SQL命令。

但无论是哪一种情况，在压缩文件解压之后，我们都能得到相应的数据文件，并利用它们完成目标任务。那怎么才能知道应该使用哪一个程序来解压文件呢？这当中最重要的线索就是文件扩展名。根据这个信息，我们可以得知文件是由哪种压缩程序创建的，进而得知文件的解压办法。

在Windows系统上，你可以在文件上点击鼠标右键，选择Properties，这时你就可以看到与文件扩展名关联的程序了。之后可以使用Open With选项来查看Windows会使用哪种程序来解压文件。

在这一节后面的内容里，我们会继续学习如何在OS X或Linux操作系统上使用命令行程序来解压文件。

(1) 使用zip、gzip和bzip2压缩文件

zip、gzip和bzip2是最为常见的压缩程序。与之对应的解压缩程序分别为Unzip、Gunzip和Bunzip2。

下表中列举了每种程序以及它们对应的压缩和解压缩命令。

	压 缩	解 压
Zip	`zip filename.csv filename.zip`	`unzip filename.zip`
gzip	`gzip filename.csv filename.gz`	`gunzip filename.gz`
bzip2	`bzip2 filename.csv filename.bz2`	`bunzip2 filename.bz2`

有时你会发现，有的文件在包含.tar后缀名的同时还包含了.gz或.bz2这样的后缀名，例如somefile.tar.gz。其他类似的情形还有.tgz和.tbz2等，例如somefile.tgz。这些文件一般都是先经过tar程序的归档处理，之后又被gzip和bzip2程序进行压缩。这是因为gzip和bzip2本身没有归档功能，它们只是压缩程序而已。因此，它们只能一次压缩一个文件（或一个归档文件）。而tar的工作才是把多个文件整合到一个独立的文件中，所以我们常常会发现这些工具总是一起使用。

其实tar程序还有个内置的功能选项，可以让我们在文件归档之后立即使用gzip或bzip2对文件进行压缩处理。要想压缩一个新创建的.tar文件，可以在tar命令的后面添加一个`z`选项，并修改它的文件名：

```
tar cvzf fileArchive.tar.gz reallyBigFile.csv anotherBigFile.csv
```

或者，你可以分两个步骤来完成这个任务：

```
tar cvf fileArchive.tar reallyBigFile.csv anotherBigFile.csv
gzip fileArchive.tar
```

这两种操作都可以创建文件fileArchive.tar.gz。

如果想要解压一个tar.gz文件，可以使用下面两个命令：

```
gunzip fileArchive.tar.gz
tar xvf fileArchive.tar
```

类似的操作也适用于bzip2文件：

```
tar cvjf fileArchive.tar.bz2 reallyBigFile.csv anotherBigFile.csv
```

如果想要解压一个tar.bz2文件，可以使用下面两个命令：

```
bunzip2 fileArchive.tar.bz
tar xvf fileArchive.tar
```

(2) 压缩选项

在做压缩和解压处理时，你应该考虑如何使用更多的功能选项来更好地完成清洗工作。

- 你是否需要在压缩文件的时候保留原始文件？默认情况下，大部分压缩程序和归档程序都会删除原始文件。如果你想在创建压缩文件的同时保留原始文件，通常需要手动添加这个选项。
- 你想把新的文件添加到一个已经存在的压缩文件里吗？大多数归档程序和压缩程序都有这样的功能选项。有时候，这种操作被称为*更新*或*替换*。
- 你是否需要对压缩文件进行加密处理，只有提供密码才能打开？其实许多压缩程序都带有这个功能。
- 在解压的时候，是否需要覆盖目录下同名的文件呢？找找有没有force这样的功能选项吧。

不同的压缩软件有不同的功能选项，你可以根据需要使用这些选项来轻松地完成文件处理工作。这在处理大量文件的时候尤为重要，可能是文件的体积大，也可能是数量大。

3. 如何选择压缩程序

这里所讲的归档与压缩的概念广泛适用于各种操作系统和各种类型的压缩文件。在大多数情况下，我们都是从不同的地方下载压缩文件，随后最为关心的是如何高效地解压这些文件。

但是，假设你需要自己一个人来完成压缩文件的创建，这时你应该怎么做？比如你需要解压数据文件，清洗文件中的数据，然后重新对文件进行压缩处理并把它转给你的同事，这种情况下你该怎么处理每一个细节？又或者是你要下载一些文件，而每个文件都提供了不同的压缩格式：zip、bz2或gz，哪种格式才是最佳选择呢？

假设我们处在一个可以使用多种文件压缩格式的操作环境中，这时可以采用下面的原则来比较不同压缩类型之间的优缺点。

其中能够影响我们选择压缩方式的关键因素有：

- 压缩和解压缩的速度
- 压缩比率（文件到底能缩减多少）
- 压缩方案的互操作性（文件容易解压吗）

● 经验之谈

gzip压缩和解压缩都比较快，并且在每个装有OS X或Linux的机器上都可以使用。但是，在Windows系统上的一些用户是没有对应的gunzip程序的。

bzip2压缩的文件比gzip和zip都要小，但压缩花费的时间也相对长一些。它广泛存在于OS X和Linux系统中。而Windows用户就比较痛苦了，除非他们装了能够应对bzip2格式的特殊软件。

zip在Linux、OS X和Windows系统上都存在，而且它的压缩和解压速度也不赖。但是压缩比率不怎么高（其他压缩程序能让文件变得更小）。即便如此，它的普遍性和较快速度（比如跟bzip2相比）还是相当有优势的。

RAR是Windows上广泛使用的归档和压缩解决方案；但是在OS X和Linux系统上只有特殊的软件才能处理这种格式，而且它的压缩速度也不像其他方案那样理想。

最后要说的是，你得根据项目的具体情况，以及受众或用户（可能是你本人、消费者或者是客户）的需求，来决定压缩方案。

2.3　数据类型、空值与编码

这一节我们将了解一下最常见的数据类型，这些类型及其变体类型都是数据科学家们在日常工作中需要频繁处理的。此外，我们还会讨论如何在数据类型之间进行安全且无信息损耗（或是至少要事先了解风险程度）的转换。

同时这一节探索空数据、空值和空白字符的神秘世界，学习各种不同的数据缺失情况，看看数据缺失会对数据分析结果有哪些负面影响。还将比较和权衡数据缺失的处理方案以及它们各自的优缺点。

由于在实际存储中很多数据都是以字符串形式保存的，所以我们还要学习如何辨识字符编码，以及真实数据中一些常见的格式。我们还会学习如何识别字符编码问题，以及如何为某个特定的数据集设置与其相匹配的字符编码。此外还会以Python代码为例，演示把数据从一种编码转

成另一种编码的过程，并会讨论这种策略的局限性。

2.3.1 数据类型

无论要清洗的数据是以文本文件形式存储，还是在数据库系统中存储，或是以其他格式存储，我们慢慢都会发现一个现象，那就是相同类型的数据看起来总是一样的：数字、日期、时间、字符、字符串，等等。在这一节中我们就要探讨一些最为常见的数据类型以及与相关的一些例子。

1. 数字类型数据

在这一部分中，我们会展示多种用于数字存储的方式和方法，其中的一些方法会让清洗和管理工作变得更加容易。与字符串和日期类型相比，数字类型更为直观，所以我们先从这个类型开启旅程，之后再学习更为复杂的类型。

(1) 整数

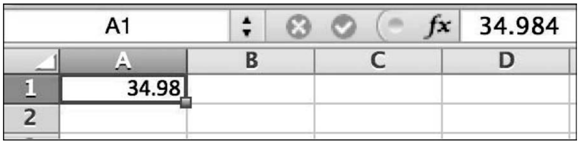
整数，可以是正数或者是负数。从整数的名字上可以看出，这种数字是没有小数点和小数部分的。在不同的存储系统中，比如在*数据库管理系统*（DBMS）里，我们可以对整数进行更多的设置，比如整数的范围是多少，是允许有符号值（正数或负数）还是只允许有无符号值（全部正数值）。

(2) 小数

在数据清洗工作中，含有小数部分的数字，比如价格、平均值、尺寸等，通常都采用小数点形式（而不是分子/分母）来表现。有时候，个别的存储系统还会提供一些数字设置规则，包括允许小数点后面有多少个数字（*小数部分的长度*），允许包含的数字总个数（*精度*）。例如数字34.984，它的小数部分长度为3，精度为5。

不同的数据存储系统允许使用不同类型的小数。例如，DBMS可以让我们在搭建数据库的时候自行决定数据存放形式，如浮点数、小数、货币。每种类型之间都稍有差别，比如在数学含义上的区别。我们需要阅读DBMS所提供的相关指南来掌握好每种数据类型，从而可以始终在数据变化时占据主动地位。DBMS供应商常会出于内存原因或其他原因多次修改某些数据类型的规范。

而在另一个方面，与存储数据的DBMS系统不一样的是，电子表格应用程序除了具有存储数据的能力之外，它还能显示数据。因此，在实际使用中我们可以用某一种数字类型的形式存储数据，而以另外一种类型的形式来显示它。但这种格式化方法有时也会引起一些歧义。下图是一个显示小数的例子。公式栏里显示的完整数字是34.984，但实际上单元格显示的是经过四舍五入之后的数字。



> 在世界上的很多地区，人们用逗号来分隔数字的小数部分和非小数部分，而不是我们习惯使用的圆点或英文句号。所以，花点时间搞清楚操作系统的区域设置是很值得的，这可以确保数据能够像我们预期的那样显示。比如在OS X中，菜单System Preferences里有一个称为Language & Region的对话框。在这里你可以根据需要修改区域设置。

与DBMS不同的是，纯文本文件没有可以用来规范数字精度的选项。文本文件也不同于电子表格，数据显示上也没什么额外的设置选项。如果文本文件里显示的数值为34.98，那么这就是全部的数字信息了。

(3) 什么时候数字不是数字类型

数字类型的数据最重要的特点就是由一系列0~9之间的数字组成，有时候也会包含小数部分。但真正的数字类型数据还有一个关键的特性，就是能够满足其最初的设计目的，参与数学计算。所以，当希望能用数据进行数学计算，或是以数字形式进行比较，又或者按数值顺序存储项目的时候，就应该选择数字类型作为数据的存储方式。每当遇到与此类似的情况，请务必确保数据都以数字类型存储。请看下面的数字列表，其中数字都是按照从小到大的顺序排列的：

- 1
- 10
- 11
- 123
- 245
- 1016

现在，再看一个地址列表，字段中的数据都是以字符串类型存储的：

- 1 Elm Lane
- 10 Pine Cir.
- 1016 Pine Cir.
- 11 Front St.
- 123 Main St.
- 245 Oak Ave.

电话号码和邮政编码（还有上面例子里的门牌号码）通常都是以数字形式存储的，但是当我们考虑把它们保存下来的时候，应该是用文本形式还是数字形式呢？先试问一下我们是否计划拿这些数据进行加减法操作，或是求平均值，又或是求标准差？如果没有这些打算的话，那以文本形式保存更为妥帖。

2.日期和时间

你可能对同一个日期的很多不同写法都很熟悉，而且有自己比较喜欢的写法。比如下面列表中的每一个小项，它们都是同一个日期的常见写法：

- 11-23-14
- 11-23-2014
- 23-11-2014
- 2014-11-23
- 23-Nov-14
- November 23, 2014
- 23 November 2014
- Nov. 23, 2014

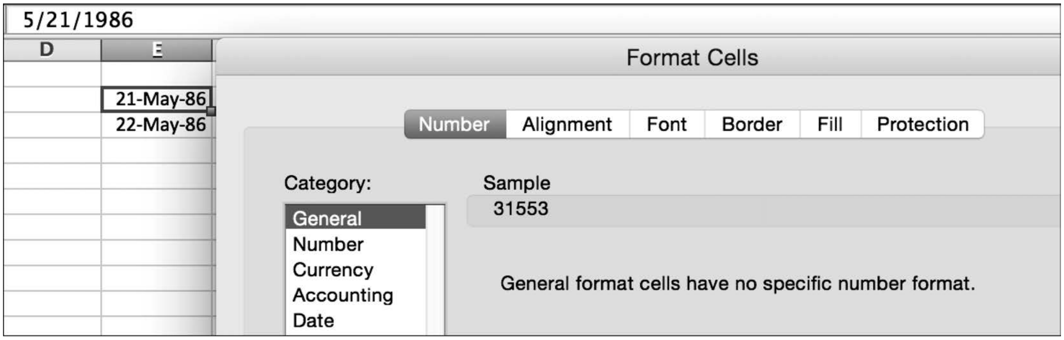
无论我们喜欢使用哪种日期书写格式，一个完整的日期都是由三部分组成：年，月，日。任何日期数据都应该能够解析出这几个部分。日期中有争议的部分一般有两处：一处是缺少日期记号和月份记号并且数字小于12，另一处则是年份信息的表现。例如，如果只看到“11-23”的话，我们会推算出日期为11月23日，因为月份不可能是23，那年份信息哪里去了？但是，如果我们看到的是“11-12”，这回是11月12日，还是12月11日呢？同样的问题，年份信息哪里去了？假如给出了年份信息，其值为38，那么它代表的是1938年还是2038呢？

大多数的DBMS都有一套专属的方法来导入日期格式的数据，并且能够使用同样的方法以日期格式导出数据。另外，这些系统还同时提供了很多函数，利用这些函数可以对日期数据进行重新格式化处理，或是提取日期中某一部分的数据。以MySQL数据库为例，它就有许多这样的日期函数，通过这些函数可以提取日期中的月份信息或是日期信息；当然，还有一些更为复杂的函数，通过它们我们可以找出一个日期处于一年中的第几个星期，或是找出一个日期处于一个星期中的第几天。请看下面的例子，我们利用第1章中提到的Enron数据集，使用SQL查询语句来计算在每年的5月12日有多少封邮件被发出，同时打印出相应的星期值：

```
SELECT YEAR(date) AS yr, DAYOFWEEK(date) AS day, COUNT(mid)
FROM message WHERE MONTHNAME(date) = "May" AND DAY(date) = 12
GROUP BY yr, day
ORDER BY yr ASC;
```

有些电子表格程序，如Excel，在内部以数字形式存储日期，但它允许用户以他们喜欢的内置格式或自定义格式来显示这些值。Excel用小数来保存自1899年12月31日之后的日期数据。想

要看到Excel中日期的内部表现，可以输入一个日期数据，并以General格式进行查看，如下图所示。Excel将1986年5月21日存储为31553。



所以，当你在不同的格式之间来回转换的时候，从斜线到中横线，或是互换月份和日期的顺序，Excel只是为你提供了不同的外部表现，但在底层的内部实现上，日期的值是从来都没有发生过变化的。

那Excel为什么保留小数部分来存储1899年之后的日期呢？难道天数不是完整的数字吗？其实，Excel存储的小数部分值是用来表现时间部分的。

E	F
formatted as a date and time	formatted as 'general'
5/21/86 12:00 AM	31553
5/21/86 12:00 PM	31553.5
5/21/86 6:00 PM	31553.75

从前面的示例截图中可以看出，日期的内部值31553被映射到午夜，而31553.5（半天时间）是中午，31553.75是下午六点。小数点后面的精度越高，我们得到的内部表现就越精准。

但是并非所有的数据存储系统都使用小数来存储日期和时间，因为它们采用的起点标准可能不同。有些系统以数字形式记录自Unix时间（国际标准时间1970年1月1日0时0分0秒）开始后所经过的秒数，以此来存储日期和时间，如果是负数，则表示Unix时间开始之前的时间。

DBMS和电子表格应用程序都支持与数字类似的日期计算。在这两种系统中，都有支持加减法以及其他日期计算的函数，比如往一个日期中添加几个星期得到一个新的日期值。

3. 字符串

字符串代表了一组连续的字符数据，其中包括常见的字母、数字、空格和标点符号，还有来自千百种自然语言中的字符和用于不同目的的特殊符号。字符串非常灵活，这一点使得它们成为最为常见的数据存储方式。此外，由于字符串几乎能够存储任何其他类型的数据（不计效率），它已成为数据通信或系统间数据移植的最廉价选择。

与数字类型数据一样，我们所使用的存储机制也提供了一些关于字符串的使用规则。例如，DBMS或是电子表格需要我们事先声明要使用的字符串长度为多少，或是要使用哪种类型的字符。还有一个重要的概念，那就是字符编码，在本章的后面我们单独拿出一节来对它进行说明。

另外，在具体的使用环境中我们还需要注意数据的长度限制。在日常使用的数据库中，通常有固定长度和可变长度两种字符串类型，它们一般被设计用来存放较短的字符串数据，所以有的DBMS提供商还额外设计了一种文本类型，专门用来存放较长的字符串数据。

> 一般情况下，许多数据科学家会扩大这个术语的范围。因为字符串类型的数据可能会变得越来越大，毫无边际，所以它们往往会发展成文本类型的数据，对它们的分析也就变成文本分析或文本挖掘。

字符串（或文本）类型的数据可以在之前谈论过的所有文件格式（分隔文件、JSON或网页）中找到，这些格式能够以字符串形式存储数据，或是为多种存储方案（比如前面提到过的API、DBMS、电子表格和纯文本文件）提供数据。但不管采用哪种存储与传递机制，在大数据、乱数据、无结构数据的处理过程中，字符串类型的数据可以算是当之无愧的主角。如果只是解析区区几百个街道地址或是对几千本书的标题进行分类，对Excel高手来说简直毫无压力。同样，如果让统计工作者或程序员从一个单词列表中找出字符出现的频度，他们也绝不会犹豫。不过，如果把字符串操作的问题提升到“从九千万条用俄语书写的电子邮件消息中抽取嵌套在里面的程序源代码”，或是“计算Stack Overflow网站上全部内容中的词汇多样性”，事情可就复杂啦。

4. 其他数据类型

数字、日期/时间和字符串是使用最为广泛的三种数据类型，但除此之外，还有许多其他来源于不同环境的数据类型。请看下面的例子。

- **集合/枚举**：如果你的数据只有几种值，你可能就需要集合类型或者枚举类型了。比如大学课程的期末成绩就是由这样的枚举类型数据组成的：{A, A−, B+, B, B−, C+, C, C−, D+, D, D−, F, W}。
- **布尔**：如果你的数据只限于两种值，要么是0/1形式，要么是true/false形式，这时你可以考虑一下布尔类型。比如数据库字段`package_shipped`的值可能为“是”和“否”，这表示包裹是否已经邮寄出去了。
- **Blob**（二进制大对象）：如果你的数据是二进制的，例如你想要以字节形式存储图片（并非图片链接），或是存储MP3音乐文件，这时你就可以考虑使用二级制大对象类型。

2.3.2 数据类型间的相互转换

数据类型间的相互转换是数据清洗工作必不可少的一部分。每当需要在字符串类型的数据上

做一些数学运算处理时，我们往往希望能够以数字类型来重新存储这些数据。又或者是碰到字符串形式的日期数据，我们希望改变它们的日期表现格式。但在正式学习数据类型转换之前，我们先讨论一个有关转换的潜在问题。

数据损耗

从一种数据类型转换到另一种数据类型的过程中，有时我们会面临着数据损耗这样的境况。通常，在目标数据类型无法保存与原始数据类型同样多的信息时，损耗就会发生。其实数据损耗并不会一直困扰我们（事实上，有些时候清洗过程是允许数据损耗存在的），但如果我们并不希望有数据损耗发生的话，这一现象就不容忽视了。其中包含的风险因素包括如下两个。

- **同种类型间的不同范围转换**：假设你有个字符串类型的字段，其长度为200个字符，现在你想把数据转到一个只能容纳100个字符的字段上。只要是超过100个字符的数据都会被丢弃或截取。这种情况也会发生在不同范围的数字类型字段上，例如从长整数类型向一般长度的整数类型转换，或是从整数类型向长度更小的整数类型转换。
- **不同精度间的转换**：假设你有一个小数列表，其中每一个值的精度都为四位数字，现在你想把数字转换成精度为两位数字的数据，或许更糟，直接转换成整数类型。这时每个数字都会被四舍五入或被截取，你将丢掉原有的精度信息。

2.3.3 转换策略

数据类型的转换策略有多种，但具体要使用哪一种取决于数据的存储位置。下面将讲述两种在数据科学清洗过程中比较常见的数据类型转换策略。

第一种策略，基于SQL的操作。这种策略适用于处理保存在数据库中的数据。我们可以使用数据库函数（这个功能几乎在每个数据库系统中都能找到），把数据加工成不同的格式，直至它们适合以查询结果的形式直接导出或是存放到另一个字段中去。

第二种策略，基于文件的操作。这种策略可以用来处理文本类型的数据文件，例如电子表格或JSON文件，我们需要把从文件中读取出来的数据再次以某种方法进行进一步加工。

1. SQL级别的类型转换

接下来将演示几个适用于采用SQL进行数据类型转换的常见案例。

(1) 例子一：解析MySQL数据并格式化为字符串

在这个例子里，我们需要回到第1章中提到的Enron数据库。与之前的例子一样，我们先查看数据表message，该表中包含一个日期字段，采用的是MySQL的`datetime`类型。假设我们想要打印出完整的月份拼写（与数字相对）、星期和时间。怎么做才能达到最佳效果呢？

目前我们手里有一条消息ID（mid）为52的记录，内容如下：

```
2000-01-21 04:51:00
```

而我们希望得到的格式则是：

```
4:51am, Friday, January 21, 2000
```


- 选择一：把`concat()`和日期时间函数结合起来使用，直接输出我们希望看到的结果，具体代码如下。但这种方式的弱点是不太容易输出a.m/p.m：

```
SELECT concat(hour(date),
':',
minute(date),
', ',
dayname(date),
', ',
monthname(date),
' ',
day(date),
', ',
year(date))
FROM message
  WHERE mid=52;
```

  结果：

```
4:51, Friday, January 21, 2000
```

  如果我们必须要包含a.m/p.m，可以使用条件语句来判断小时部分的值，如果小于12则打印“a.m”，反之打印“p.m”：

```
SELECT concat(
  concat(hour(date),
    ':',
    minute(date)),
  if(hour(date)<12,'am','pm'),
  concat(
    ', ',
    dayname(date),
    ', ',
    monthname(date),
    ' ',
    day(date),
    ', ',
    year(date)
  )
)
FROM message
WHERE mid=52;
```

  结果：

```
4:51am Friday, January 21, 2000
```

> MySQL的日期和时间函数，如day()和year()，它们的开发文档信息位于http://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html；字符串函数，如concat()，其开发文档位于http://dev.mysql.com/doc/refman/5.7/en/string-functions.html。如果使用的是别的数据库管理系统，我们也可以找到有着类似功能的函数。

- 选择二：使用MySQL提供的更为高级的date_format()函数。这个函数能够根据我们提供的一系列字符串说明符来对日期进行格式化处理。MySQL的开发文档对这些说明符做了非常详尽的解释。下面例子中演示的代码就是把日期类型数据转成我们期望的格式：

```
SELECT date_format(date, '%l:%i%p, %W, %M %e, %Y')
FROM message
WHERE mid=52;
```

结果：

```
4:51AM, Friday, January 21, 2000
```

这已经非常接近我们希望的结果了，而且代码要比选择一中的精炼得多。唯一有偏差的地方就是a.m./p.m都是大写的。如果我们想要小写的形式，可以像下面这样修改代码：

```
SELECT concat(
  date_format(date, '%l:%i'),
  lower(date_format(date,'%p ')),
  date_format(date,'%W, %M %e, %Y')
)
FROM message
WHERE mid=52;
```

结果：

```
4:51am, Friday, January 21, 2000
```

(2) 例子二：从字符串类型转换到MySQL的日期类型

在这个例子中，我们需要使用Enron数据库下的一张新表，这个表的名字是referenceinfo。它用于保存由其他消息所引用的消息。举例来说，该表中的第一条记录有一个值为2的rfid字段，其中包含了第79条记录所引用的电子邮件内容。该字段类型为字符串，数据内容如下（部分截取）：

```
> From: Le Vine, Debi> Sent: Thursday, August 17, 2000 6:29 PM> To: ISO Market
Participants> Subject: Request for Bids - Contract for Generation Under Gas>
Curtailment Conditions>> Attached is a Request for Bids to supply the California ISO
with> Generation
```

内容挺乱的！先让我们提取首行中的日期并把它转成MySQL的日期类型吧，因为这种类型的数据才适合插入数据表或是参与日期计算。

要完成这项任务，需要用到MySQL的内置函数`str_to_date()`。这个函数与前面讨论的`date_format()`有点像，只是它用于逆向处理。下面的查询语句先是找到单词`Sent:`，并继续查找后续的字符直到>符号为止，然后把得到的字符串转换成MySQL的`datetime`类型：

```
SELECT
str_to_date(
  substring_index(
    substring_index(reference,'>',3),
    'Sent: ',
    -1
  ),
  '%W,%M %e, %Y %h:%i %p'
)
FROM referenceinfo
WHERE mid=79;
```

结果：

```
2000-08-17 18:29:00
```

现在我们已经有了`datetime`类型的数据了，可以选择把它直接插入一个新的MySQL字段，或是利用更多的日期函数来对其进行计算。

(3) 例子三：把MySQL字符串类型的数据转换成小数

在这个例子中，我们需要考虑如何把文本中的数字转换成适用于计算的数据格式。

假设我们要从安然公司的某些电子邮件消息中提取每桶（缩写为bbl）原油的价格。首先需要做的是编写一条查询语句，从某个发件人所发出的邮件中查找含有`/bbl`字样的字符串，之后再向前查找美元符号并提取与之关联的小数形式的数据。

下面的样例取自Enron数据库中的message表，消息ID（mid）为270516，先让我们看看字符串中的数字是什么样子的：

```
March had slipped by 51 cts at the same time to trade at $18.47/bbl.
```

下一步我们可以使用MySQL命令提取字符串并执行数字类型的转换操作：

```
SELECT convert(
         substring_index(
           substring(
             body,
             locate('$',body)+1
           ),
           '/bbl',
           1
         ),
         decimal(4,2)
      ) as price
```

```
FROM message
WHERE body LIKE "%$%" AND body LIKE "%/bbl%" AND sender =
'energybulletin@platts.com';
```

引入`WHERE`子句可以确保我们得到的消息包含原油价格。

函数`convert()`与`cast()`的功能相似。大多数现代数据库系统都拥有把某种数据类型转成数字的功能。

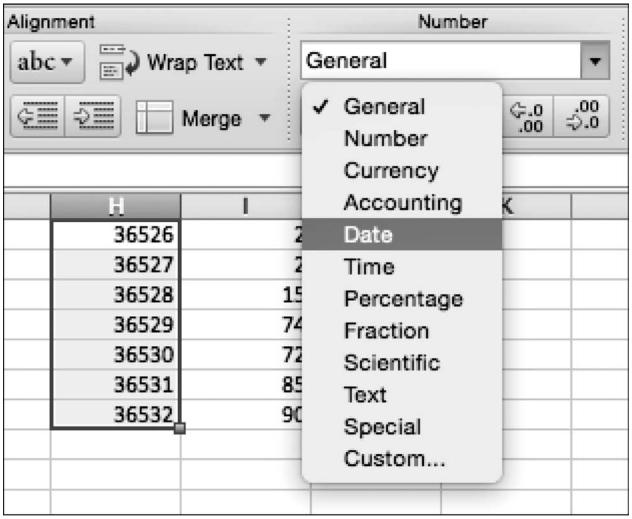
2. 文件级别的类型转换

在这一节中，我们将演示一些在文件级别上操作数据类型的常见案例。

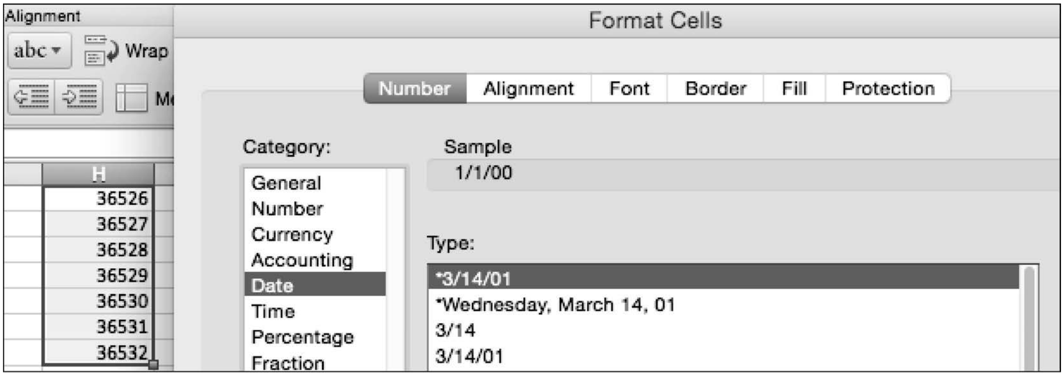
> 这一节讨论的内容只适用于采用隐式类型结构的文件类型，例如，电子表格和半结构化的JSON数据。这里不包括使用分隔技术（只有文本信息）的纯文本文件，因为在这种类型的文件中，所有的数据都是文本数据。

(1) 例子一：Excel中的类型发现与转换

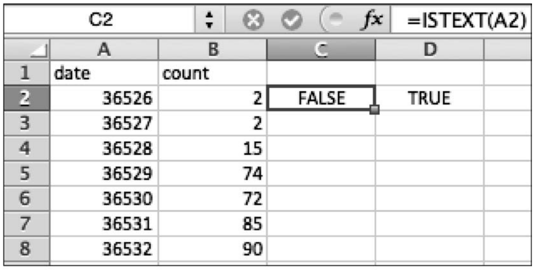
可能大多数人都知道，在Excel和类似的电子表格程序中，可以使用格式化菜单来进行数据类型转换。一般的做法是选择想要修改的单元格，然后使用ribbon上的下拉菜单进行操作。



如果这并不能满足你的需求，还可以使用一个名为Format Cells的对话框，它可以通过格式化菜单打开。利用对话框中提供的选项，可以对转换处理的输出内容进行更为细粒度的控制。



其实在Excel中还有两个较少有人知道的函数，`istext()`和`isnumber()`，它们在格式化数据方面用处也非常大。

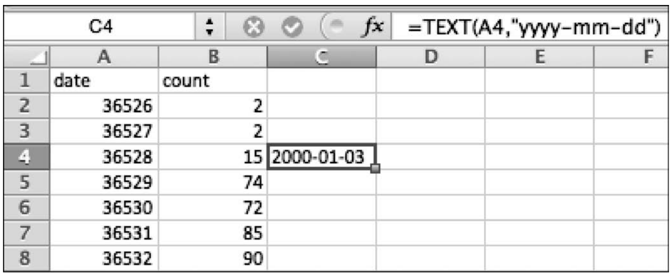


C2　=ISTEXT(A2)

	A	B	C	D
1	date	count		
2	36526	2	FALSE	TRUE
3	36527	2		
4	36528	15		
5	36529	74		
6	36530	72		
7	36531	85		
8	36532	90		

这两个函数适用于任意一个单元格，`istext()`会依据单元格中数据是文本类型或非文本类型而返回TRUE或FALSE，而`isnumber()`则会依据数据是否为数字给出类似的结果。在与条件格式化特性配合使用时，这两个公式可以帮你在数据量不多的情况下定位错误数据，或是不正确的输入数据。

另外，Excel还提供了一些不需要使用菜单就能完成从字符串类型到其他数据类型转换的函数。下图演示了如何用`TEXT()`函数把数字类型的日期转换成格式为yyyy-mm-dd的字符串类型日期。在公式栏，只需输入`=TEXT(A4,"yyyy-mm-dd")`，数字`36528`就会被转换成`2000-01-03`。这样就能得到字符串类型的日期数据了。



C4　=TEXT(A4,"yyyy-mm-dd")

	A	B	C	D	E	F
1	date	count				
2	36526	2				
3	36527	2				
4	36528	15	2000-01-03			
5	36529	74				
6	36530	72				
7	36531	85				
8	36532	90				

(2) 例子二：JSON中的类型转换

JSON作为一种基于文本的半结构化类型，并没有提供太多的格式化选项和数据类型。回想一下本章前面对JSON的描述，每个JSON对象都是以“名–值”对为基础构建出来的。唯一可以应用格式化操作的就是“名–值”对中的值部分，其类型可以是字符串、数字或列表。虽然JSON对象可以通过手工的方式来构建，比如在文本编辑器中直接输入，但是在大多数情况下，我们都是使用编程的方式来构建JSON对象的，要么从数据库中输出JSON，要么写一个小程序把纯文本文件转换成JSON。

如果我们设计的JSON对象构建程序有缺陷，那该怎么解决呢？比方说程序总是生成字符串类型数据，而并非我们想要的数字类型数据。这种情况在实际的工作中确实会存在，而且它可能会导致那些使用JSON的程序产生意想不到的结果。这里我们以下面的程序为例，先使用一段简单的PHP代码生成JSON数据，随后再利用这些JSON数据在D3程序中处理，最终生成一个图形。

PHP代码所生成的JSON数据都是直接来自数据库的。程序代码先是连接到Enron数据库，然后利用构建好的查询语句开始检索数据库，并把检索结果逐条放到数组中去，最后进行编码操作把字符串值转换成“名–值”对。代码的目的就是生成一个包含日期和数据个数的列表，日期和数据个数与第1章中使用的一样。

```
<?php
// 连接到数据库
$dbc = mysqli_connect('localhost','username','password','enron')
       or die('Error connecting to database!' . mysqli_error());

// 这里使用的查询语句与第1章按照日期进行分组的查询语句一样
$select_query = "SELECT date(date) AS dateSent, count(mid) AS numMsg FROM message GROUP
BY 1 ORDER BY 1";
$select_result = mysqli_query($dbc, $select_query);
// 查询失败则终止处理
if (!$select_result)
       die ("SELECT failed! [$select_query]" .  mysqli_error());
// 构建一个用于打印json格式数据的新数组
$counts = array();
while($row = mysqli_fetch_array($select_result))
{
     array_push($counts, array('dateSent' => $row['dateSent'], 'numMsg'   =>
$row['numMsg']));
}
echo json_encode($counts);
?>
```

> 这里要注意的是json_encode()函数需要5.3版本的PHP或是更高级的版本，并且需要使用第1章中搭建的Enron数据库。

结果数据的问题就是所有的值都是字符串类型——由于PHP把数字类型的numMsg也加上了

双引号，所以JSON也把这个值当成字符串处理：

```
[
  {"dateSent":"0001-05-30","numMsg":"2"},
  {"dateSent":"0001-06-18","numMsg":"1"}
]
```


为了使PHP函数可以真正地用于处理数字类型的数据，而不是简单地把所有数据都当作字符串类型对待，我们需要在数据打印到屏幕前做些特别的转换。其实只要在调用`json_encode()`的时候像下面那样稍微做一下调整就可以：

```
echo json_encode($counts, JSON_NUMERIC_CHECK);
```

现在JSON结果中的`numMsg`是数字类型了：

```
[
  {"dateSent":"0001-05-30","numMsg":2},
  {"dateSent":"0001-06-18","numMsg":1}
]
```

PHP还有一些类似的功能可以把大的整数类型转换成字符串类型。这种功能在处理超大数字的时候可以派上用场，比如在需要把session中的值或是Facebook、Twitter用户ID保存成字符串数据的时候。

还是上面的PHP代码，假如我们没有对生成的JSON结果进行调整的话——再来看一个例子，假设我们通过API来获取JSON数据结果，并且显示内容没有经过任何调整——就不得不在JSON对象构建之后对它进行转换。如果碰到这种情况，我们可以在使用D3时通过JavaScript代码，利用+操作符来迫使目标数据从字符串类型转换成数字类型。具体代码如下所示，我们先是读取输出的JSON内容，然后构建图形。字段`numMsg`的值已经从字符串类型被强制转换成数字类型：

```
d3.json("counts.json", function(d) {
  return {
    dateSent: d.dateSent,
    numMsg: +d.numMsg
  };
}, function(error, rows) {
  console.log(rows);
});
```

2.3.4 隐藏在数据森林中的空值

在这一节，我们要把我们的勺子伸到混有零、空值和null值的大杂烩汤里去了。

这三者之间有什么分别呢？请听我慢慢道来。假想在一个理想的厨房里，你有一个高级的烤炉，炉子上面是一锅香浓的老汤。这时你的副厨师长问你：“锅里还有多少汤？”请看下面三种情况的回答。

(1) 在一开始你就发现，炉子上根本没有锅。要是这样的话，“锅里有多少汤？”这个问题就没法回答了。连锅都没有！答案不是正数，也不是零，甚至连空都不是。答案是NULL。

(2) 经过几个小时之后，食材已经洗净并且也切好了，你往锅里瞅了一眼，刚好看到有三公升的老汤。太好了，你现在可以回答这个问题了。这种情形下，答案应该是3。

(3) 经过午餐的高峰期之后，你又往锅里看了一眼，发现没有汤了，甚至连一滴汤水都没有。你看了又看、查了又查，问题“锅里有多少汤？”的答案只能是零。这时，你顺势把锅丢进了洗碗池。

(4) 晚饭前，你从水池中取出洗刷干净的汤锅。正当你经过厨房的时候，副厨师长又发问了：“锅里有什么？”这时锅里是空的，里面什么东西都没有。但要注意的是，这次答案不是零，因为他要的答案是不可以用数字表达的。当然，答案也不是NULL，因为锅还在，你再次仔细地看看，答案就是空。

不同的数据环境和编程环境（DBMS、存储系统和编程语言）在对待零、空值和NULL这三个概念时稍有不同。这三种类型的值不是在所有的环境下都有着明确的区别。因此，在这一章中我们只讨论比较通用的内容，在列举各种示例的时候会指明具体的应用环境。当我们每次说null、空值或零的时候，请记得指出每个值的使用环境，这很重要。你在工作中遇到这些值的时候，一定得搞清楚它们所代表的含义。

> 在使用Oracle数据库的时候，请多加小心，因为Oracle中的空值、空白和NULL不同于其他系统。在读完这一节内容之后，具体细节还请参考Oracle的数据库文档。

1. 零

首先，也是最重要的事。在零、空值和NULL中，最容易处理的就是零值数据。零是可测量的数字，在数值系统中是有意义的。我们可以拿零值做排序（它出现在1、2、3…这些数字之前），还可以拿其他数字来跟零做比较（–2、–1、0、1、2、3…）。此外，零还可以参与数学计算（但有种例外的情况，零不可以做除数）。

作为一个合规的数字，零只有在数字类型中才可以发挥它最大的意义。而以字符串类型表现的零就没有这么多功能，因为它最终要么被解析成数字0，要么被解析成字符0，但这些往往都不是我们对零值的期望。

2. 空

与零相比，空值的处理要难一些，它们会在不同的情况下产生不同的含义，以字符串处理为

例。假设我们有一个用于表现中间名（middle name）的属性。可是我没有中间名，所以我得把这个字段放空。（囧事：我妈妈至今还记得我的幼儿园毕业证书的样子，当时我迫不及待地在上面编了个中间名，因为我实在不好意思跟老师讲我连个中间名都没有。）这时就算是使用空格字符或连字符来表现空值也不会有更多的意义。而且空格跟空值根本就不是一回事。表现空字符串的正确方式就是把它“留空”。

在CSV或分隔文件中，空值看起来就像下面的内容一样——注意后面的两条记录，它们的“favorite color”字段都为空：

```
First name,birth date,favorite color,salary
"Sally","1971-09-16","light blue",129000
"Manu","1984-11-03","",159960
"Martin","1978-12-10","",76888
```

而在往MySQL数据库中`INSERT`数据的时候，则使用下面的代码插入“Manu”记录：

```
INSERT INTO people (firstName, birthdate, faveoriteColor, salary)
  VALUES ("Manu","1984-11-03","",159960);
```

而对于半结构化的数据格式，如JSON，有些时候也是可能出现空对象和空字符串的。请看下面的例子：

```
{
  "firstName": "Sally",
  "birthDate": "1971-09-16",
  "faveColor": "",
  "pet": [],
  "job": {
    "jobTitle": "Data Scientist",
    "company": "Data Wizards, Inc.",
    "salary":129000
  }
}
```

在这里，我们拿走了Sally的宠物，并把她的“favorite color”字段置为空字符串。

- 空白

请注意，`" "`（两个双引号中间夹着一个空格字符，有时候也称空白，但空格的说法可能更为恰当一些）不等同于`""`（彼此紧挨着的两个双引号，有时候也称空白，但空值的说法更为恰当）。请看下面两个MySQL的`INSERT`语句，注意比较它们之间的不同之处：

```
-- this SQL has an empty for Sally's favoriteColor and a space for Frank's
INSERT INTO people (firstName, birthdate, faveoriteColor, salary)
VALUES ("Sally","1971-09-16","",129000),
       ("Frank","1975-10-23"," ",76000);
```

除了空格外，有时其他不可见字符也会被误当成空或空白来解析，这样的字符有制表符、回

车、换行符等。在使用这些字符时还是需要仔细一些的，要是有什么疑问，可以使用本章前面介绍的方法来查看这些不可见字符。

3. null

如果你查一下词典对null的解释，答案很有可能就是零。但千万不要被误导了。在实际的处理中，有很多关于NULL的定义。对于我们来说，NULL并非意味着空无一物；实际上，它的出现表明连空值都缺席了。

那它和空到底有什么分别？首先，空可以等价于空值，就像空字符串的长度为0一样。可以想象得出，空其实是一个实实在在的值，可以针对它做一些比较操作。但是NULL就不一样了，NULL不等于NULL，而且NOT NULL也不等于NOT NULL。有人就曾建议说，我们应该像念咒那样牢牢记住，NULL不等于任何值，甚至是它本身。

只有在不希望出现任何数据的情况下才应该使用NULL。这就跟我们不想把汤锅放到炉子上一样！

(1) 为什么中间名的例子可以使用“空”而不能使用NULL？

问得好。回想一下前面汤锅的例子，如果我们提出了问题并且得到的答案是空，那么这种情况与从未得到过答案（NULL）的情况完全不同。如果你问我的中间名是什么，我告诉你说我没有中间名，这时的数据应为空。但如果你只是不清楚我是否有中间名，那么数据则为NULL。

(2) 在清洗数据的时候总是用零来替代空或null会有用吗？

或许你还记得第1章中的电子邮件例子吧，那时我们讨论了怎么往条形图中添加缺失的数据，如何利用电子表格程序为缺少数据的日期自动填充零值数据。虽然我们没有真的在那些日期上挨个计算电子邮件的个数，但图形还是会补充填写缺失的数据，直接把它们当成零对待。这样做可以吗？嗯，很难说。如果我们敢保证电子邮件系统是实时的，并且日期信息也都准确无误，那么我们当然可以很自信地说我们已经掌握了发出的全部邮件，这些日期的统计数量完全可以估算为零。但是，如果使用是第1章提到的Enron数据库中的电子邮件，我们会非常明确事实不是这样的。

另外还有一种可以用零来取代空的情况，那就是使用虚拟日期或是日期片段。举个例子来说，你知道月份和年份信息，但不知道具体是哪一天，这时你就必须在所有的日期字段上插入数据。使用数据2014-11-00有可能会满足要求。但你应该把这个操作用文档记录下来（参考1.3节），因为只有这样才有可能在六个月之后，清楚地了解当时你做了什么，为什么这么做！！

2.3.5 字符编码

在早期的计算中，所有的字符串类型数据都是采用128个不同的符号构建而成的。这种早期的编码系统被称为美国标准信息交换代码（ASCII），它被广泛地应用于英文字母中。这128个字符包含a-z、A-Z、0-9、标点符号、空格以及一些如今不太常用的电传代码。如果在我们的数据科学厨房里使用这种编码系统，就好比加工现成的冷冻快餐。这的确很廉价，但也没有什么工艺和营养，不要指望这样的东西能得到客户的认可。

在20世纪90年代早期，可变长度的编码系统被提出并实现了标准化，这就是现在人们所说的UTF-8。这种可变长度方案可以容纳更多的自然语言符号和数学符号，并为未来的变化提供了更为广阔的发展空间。（这些符号集合统称为Unicode。Unicode符号编码称为UTF-8。）现在还有一种UTF-16的编码，每个字符至少占用两个字节。在编写本书的时候，UTF-8依然是Web中的主流编码格式。

而在本书中，我们主要关注的是如何把以某种编码格式存在的数据转换成另外一种编码格式。与此有关的一些示例场景如下。

- 有一个MySQL数据库，它是以某种简单的编码格式创建的（比如采用MySQL默认的256位Latin-1字符集），其中以Latin-1格式存放了一些UTF-8数据，而现在你希望把整个表改成UTF-8编码格式。
- 在Python 2.7程序里面有一些只适用于ASCII编码的函数，但现在我们迫切需要用它们处理使用UTF-8编码的文件或字符串。

在这一节中，我们将通过几个简单的示例来演示上面的场景。但在实际的工作中，你遇到的字符编码问题绝不局限于此，这里只是抛砖引玉而已。

1. 例子一：从MySQL数据中找出多字节字符

假设有一列数据，我们非常好奇其中有多少值是以多字节格式编码的。如果字符是以多字节格式进行编码的，我们可以通过比较它们的字节长度（需要使用`length()`函数）和字符长度（需要使用`char_length()`函数）来把它们找出来。

> 下面的例子里使用了MySQL所支持的MyISAM版本数据库，关于这个版本的文档信息请参考http://dev.mysql.com/doc/index-other.html。

默认情况下，MySQL中MyISAM版本的test数据库采用的是`latin1_swedish_ci`字符编码。所以，如果我们查询含有特殊字符的国家名字，得用下面的语句才能看到想要查询的Côte d’Ivoire：

```
SELECT Name, length(Name)
```

```
FROM Country
WHERE Code='CIV';
```

运行结果显示的国家名字是Côte d’Ivoire，总长度15，但它编码后的内容则是CÃ™te dÃIvoire。除此以外，在其他一些字段上也有类似的奇怪编码。为了修正这个问题，需要使用下面的SQL命令把国家名字这一字段的默认编码调整为`utf8`：

```
ALTER TABLE  Country
  CHANGE Name `Name` CHAR(52)
  CHARACTER SET utf8
  COLLATE utf8_general_ci
  NOT NULL DEFAULT  '';
```

现在可以清空数据表重新插入239个国家：

```
TRUNCATE Country;
```

现在所有的国家名字都采用UTF-8重新编码。我们可以用下面的SQL来测试下结果，看看是否有国家采用了多字节字符来展现它们的名字：

```
SELECT *
  FROM Country
  WHERE length(Name) != char_length(Name);
```

结果显示Côte d’Ivoire和法属岛屿Réunion都是多字节字符。

如果你无法访问world数据库，或是别的什么数据库，请参考下面的例子。你可以运行下面的MySQL查询命令来进行多字节字符比较：

```
SELECT length('私は、データを愛し'), char_length('私は、データを愛し');
```

在这个例子中，日语字符长度为9，而编码长度实际为27。

这项技术可以用于检测数据的字符集——当你面对大量数据而无暇逐条检查的时候，多么希望只用一条SQL语句就轻松地找出那些含有多字节的记录，并以此为依据定制接下来的清洗计划。这时你只需要执行上面的命令就可以找到当前使用多字节编码的数据。

2. 例子二：找出MySQL中以Latin-1编码存储的Unicode字符以及其等价的UTF-8编码形式

下面示例中的代码使用了`convert()`函数和`RLIKE`操作符打印出保存在MySQL数据库中并且以Latin-1格式编码的Unicode字符串。如果你的MySQL数据库也恰巧有Latin-1编码的字段的话，也可以使用这段代码，其实直至今日Latin-1编码仍旧是大部分MySQL数据库的默认编码（自MySQL 5起就一直这样）。

在这段代码中，我们使用了面向公众开放的电影与影评数据库Movielens。整套数据集广泛地遍布在互联网的许多网站上，包括它的原始项目网站：http://grouplens.org/datasets/movielens/。另外还有一个用于创建该数据库的SQL方案链接：https://github.com/ankane/movielens.sql。如果想

顺利地运行这里所演示的例子，请访问https://github.com/megansquire/datacleaning/blob/master/ch2movies.sql，获取由本书作者准备的`CREATE`和`INSERT`命令语句。通过这些命令语句，你很容易创建出演示用的数据表并完成下面的示例代码。

```
SELECT convert(
  convert(title USING BINARY) USING latin1
  ) AS 'latin1 version',
convert(
  convert(title USING BINARY) USING utf8
) AS 'utf8 version'
FROM movies
  WHERE convert(title USING BINARY)
    RLIKE concat(
    '[',
    unhex('80'),
    '-',
    unhex('FF'),
    ']'
  );
```

从下面的截图中可以看到上面的命令在phpMyAdmin中的运行结果，这里只展示Movielens数据库movies表的前三部影片信息，它们的title字段均采用Latin-1编码。

latin-1 version	utf-8 version
City of Lost Children, The (CitÃ© des enfants perd...	City of Lost Children, The (Cité des enfants perdu...
MisÃ©rables, Les (1995)	Misérables, Les (1995)
Happiness Is in the Field (Bonheur est dans le prÃ...	Happiness Is in the Field (Bonheur est dans le pré...

> 有什么办法能把一个已经存在的数据库转成UTF-8格式呢？
>
> 由于UTF-8格式在Web中的应用非常普遍，并且它在精准地传递用世界各地的自然语言的信息方面起着重要的作用，我们强烈建议你使用UTF-8编码创建数据库。假如从一开始就使用UTF-8编码，我们工作的容易程度会远远超出我们的想象。
>
> 但是，如果你的表已经使用了非UTF-8编码，不过还尚未填充数据，你最好还是把它改成UTF-8编码并重新把其中每一个字段的字符集改成UTF-8编码。接下来你就可以用UTF-8编码格式插入数据了。
>
> 最棘手的情况就是面对大量采用非UTF-8编码的数据，你需要把它们转换成UTF-8。这时候，你得先做一些计划。针对不同情况要采取不同的方法，具体取决于你是否可以不使用查询命令，或是要调整的字段和表的数量多少。在做转换计划时，你应该根据具体的数据库系统阅读相关文档。比如在MySQL数据库中执行转换时可以采用几套不同的方案，既可以用`mysqldump`，又可以用`SELECT`、`convert()`和`INSERT`。你得从这些方案中选出一个最适合当前数据库系统的。

3. 例子三：处理文件级别的UTF-8编码

有时你可能需要调整一下代码来处理文件级别的UTF-8数据。假设有一个用于收集并打印网页内容的小程序。如果大部分网页都采用UTF-8编码，我们就得在程序内部做好相应的处理工作。但不幸的是，许多编程语言还需要做一些额外的调整才能处理UTF-8数据。请看下面的Python 2.7程序，它使用了Twitter API抓取10条推文数据并把这些数据写入到文件中：

```
import twitter
import sys

####################
def oauth_login():
    CONSUMER_KEY = ''
    CONSUMER_SECRET = ''
    OAUTH_TOKEN = ''
    OAUTH_TOKEN_SECRET = ''
    auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                               CONSUMER_KEY, CONSUMER_SECRET)
    twitter_api = twitter.Twitter(auth=auth)
    return twitter_api
###################

twitter_api = oauth_login()
codeword = 'DataCleaning'
twitter_stream = twitter.TwitterStream(auth=twitter_api.auth)
stream = twitter_stream.statuses.filter(track=codeword)

f = open('outfile.txt','wb')
counter = 0
max_tweets = 10
for tweet in stream:
    print counter, "-", tweet['text'][0:10]
    f.write(tweet['text'])
    f.write('\n')
    counter += 1
    if counter >= max_tweets:
        f.close()
        sys.exit()
```

> 为了拿到上面脚本需要用到的密钥和令牌，得先设置好Twitter认证才可以，请不要对配置细节过分担心。你参考https://dev.twitter.com/apps/new简单地做一些设置就能让脚本工作，或是研究一下第10章中的Twitter挖掘案例。第10章会讲解Twitter开发者账号的全部设置过程，并详尽地描述推文数据的采集过程。

这个小程序会找出以DataCleaning为关键字的最近的10条推文记录。（之所以选择这个关键词，是因为我最近在以这个词为主题的标签下发布了几条全由表情符号和UTF-8字符组成的推文，我确信查询结果的前10条记录会产生与预期一样的结果。）但是在用上面的Python代码把这

些推文数据保存成文件的时候，程序抛出了下面的报错：

```
UnicodeEncodeError: 'ascii' codec can't encode character u'\u00c9' in position 72:
ordinal not in range(128)
```

原因是`open()`函数不能用来处理UTF-8字符。可以采用两种方法来修复这个问题：过滤UTF-8字符，或是改变文件写入的方式。

(1) 方法一：过滤UTF-8字符

如果决定使用过滤字符这个方法，我们得明白这样做可能会损失一些比较有用的数据。本章前面已经讨论过，数据损耗是一件很麻烦的事情。不过，如果我们已经充分考虑过后果并坚持过滤掉这些字符的话，需要对之前的`for`循环做如下修改：

```
for tweet in stream:
    encoded_tweet = tweet['text'].encode('ascii','ignore')
    print counter, "-", encoded_tweet[0:10]
    f.write(encoded_tweet)
```

这段新代码将冰岛语书写的推文内容从原来的：

```
Ég elska gögn
```

改成：

```
g elska ggn
```

从文本分析的角度来讲，这句话已经失去了原有的意义，因为字母g和ggn根本就不是单词。所以说，采用这种方式来清洗推文里的字符编码并非最佳方案。

(2) 方法二：以UTF-8字符写入文件

还有一种选择就是在打开文件的时候使用支持UTF-8的`codecs`或`io`库。在文件的上方添加一行导入编码库的代码，然后像下面这样操作即可：

```
f = codecs.open('outfile.txt', 'a+', 'utf-8')
```

参数`a+`表明在文件已经创建的情况下，我们希望继续往文件中追加数据。

此外，我们还可以换一种方式，在程序的顶部引入`io`库，然后使用它提供的`open()`函数，这个函数允许传入一个指定的编码，代码如下：

```
f= io.open('outfile.txt', 'a+', encoding='utf8')
```

我们可以使用Python 3吗？为什么例子还使用2.7版本呢？

Python 3在处理UTF-8编码方面确实要比Python 2.7容易得多，这是个不争的事实。如果你的开发环境是Python 3，那么你完全可以使用它。我个人更喜欢使用Enthought Canopy来做数据分析、数据挖掘和教学工作。Python有许多发行版本——Enthought是其中之一——是为2.7版本编写的，并且在一段时间内不大可能升迁到Python 3。原因是Python 3版本对语言内部做了相当大的调整（比如我们一开始说的UTF-8原生处理能力），这意味着有许多重要的有用的包不得不重新编写以与新版本兼容。重写过程是相当耗费时间的。关于这个问题的更多信息，请参考https://wiki.python.org/moin/Python2orPython3。

2.4　小结

这一章涵盖了许多基本概念，本书后面讲数据清洗时都会用到。这些技术中，有的很简单，有的则很另类。我们已经学习了文件格式、压缩、数据类型、文件级别的字符编码和数据库级别的字符编码。在接下来的一章中，我们将再学习两个新的数据清洗工具：电子表格和文本编辑器。

第 3 章

数据清洗的老黄牛——电子表格和文本编辑器

在设计家庭厨房的时候，最典型的一种布局设计就是使用经典的三角区，每个角落分别放置冰箱、水槽和炉灶。而在数据清洗的厨房里，我们也有几样不可缺少的设施，其中包括低调的电子表格应用和文本编辑器。虽然它们看起来平淡无奇并且常常被忽视，不过一旦掌握它们的功能特性，我们的清洗工作将变得又方便又快捷。在第2章中，通过数据类型和文件类型的学习，我们已经简单地掌握了这两种工具的使用方法，而在这一章，我们将深入学习以下内容。

- 学会使用Excel和电子表格应用程序来进行数据操作，具体内容包括文本分列，字符串的拆分与拼接，异常数据的查找与格式化，数据排序，从电子表格向MySQL数据库导出数据，利用电子表格来生成SQL语句。
- 学会使用文本编辑器实现数据的自动抽取与操作，并把它们转换成对我们更为有用的格式，具体内容包括使用正则表达式完成数据的查找与替换，修改数据的行首和行尾信息，基于列模式的编辑。
- 在小型项目中同时使用这两种工具来完成数据清洗任务。

3.1　电子表格中的数据清洗

电子表格在数据清洗方面的功能主要体现在两个方面：一是它可以将数据组织成列和行，二是它的内置函数。在这一节中，我们将学习如何最大限度地发挥电子表格的功能来满足数据清洗任务的需求。

3.1.1　Excel 的文本分列功能

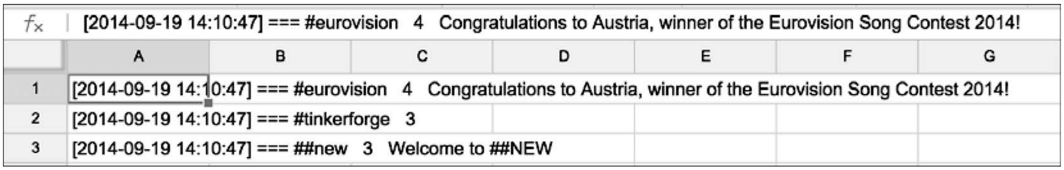
由于电子表格是采用行和列的设计方案保存数据的，所以在使用这个工具进行数据清洗时，我们要做的第一件事就是整理好数据。例如，在把大量数据粘贴到Excel或Google Spreadsheets的时候，这些软件工具首先会尝试查找分隔符号（如逗号或制表符），然后根据相应的分隔符号把

数据拆分成不同的列（请参考第2章来回顾分隔数据相关的内容）。但在有些时候，电子表格可能找不到分隔符号，这时，我们就必须人为地提供更多指导来把数据拆分成不同的列。请看下面的文本数据片段，这些数据是从Freenode的几千个在线聊天主题列表中摘取的：

```
[2014-09-19 14:10:47] === #eurovision  4  Congratulations to Austria, winner of the Eurovision Song Contest 2014!
[2014-09-19 14:10:47] === #tinkerforge  3
[2014-09-19 14:10:47] === ##new  3  Welcome to ##NEW
```

> 要想得到IRC聊天服务器上的一份聊天频道和主题数据，可以使用`alis`命令。这个命令可以作为`/query`或`/msg`的一部分被发出，至于到底会使用哪一个，由服务器设置决定。在Freenode上，命令`/msg alis *`会生成频道列表。关于IRC的更多信息请参考https://freenode.net/services.shtml。

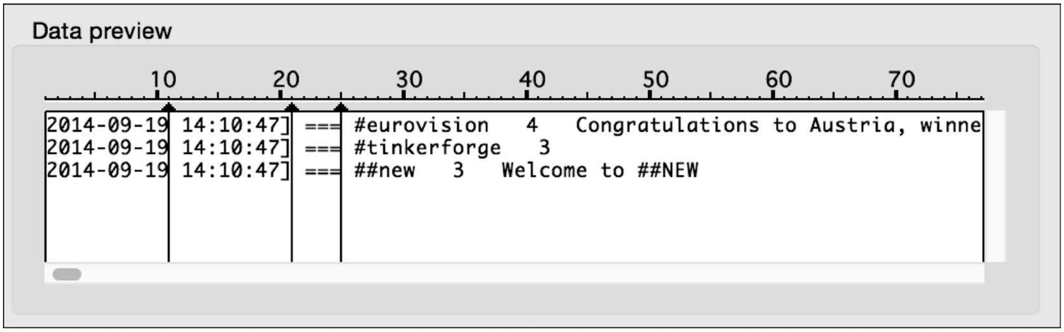
单凭肉眼我们就可以看出，处于最前面的数据是时间戳类型，紧接着的是`===`，然后是`#`和频道名称，频道名称后面是一个数字（表示在构建频道列表时该频道中有多少用户），最后是频道的描述信息。但是，当我们把这些数据粘贴到Excel或是Google Spreadsheets后，是不可能自动地得到对应的列信息的。虽然行向数据解析没有问题，但由于数据的分隔符号有很多不一致的地方，所以列向数据无法自动拆分。下面的截图中的内容是数据在Google Spreadsheets中的表现。从高亮的单元格A1中可以看出，整行数据都是置于函数栏的，这表明该行数据在粘贴后全部处于单元格A1中。



那怎么才能在电子表格中比较容易地创建列数据，让每个独立的数据都放在它们自己所对应的列中呢？只有解决了这个问题，我们才有机会完成更多任务，比如求得频道中的用户平均数，或按照频道名称来对数据排序。但就目前的情形来看，面对这些复杂的字符串文本数据，我们是无法轻松地搞定数据排序或是利用函数进行数据加工的。

想要解决这个问题，办法之一就是使用Excel的文本分列向导，将数据拆分成可以识别的几大块，之后再重新组织数据并按照需求剔除不需要的字符。具体的操作步骤如下。

(1) 选中列A，然后开启文本分列向导（位于数据菜单中）。在向导的第一个步骤中选择固定宽度，在第二个步骤中双击绘制在描述字段上的分割线。下面就是操作后的截图，只有前几列被拆分，多余的分割线已经被移除。



Excel的宽度固定拆分效果

现在结果数据看起来应该与下图相同。前面的三列数据看起来还不错，但宽度固定分隔方法不适用于D列中的用户数量和频道名字。这是因为与前面几列数据不同，频道名称的长度无法预测。



	A	B	C	D	E	F	G	H	I	J
1	[2014-09-19	14:10:47]	===	#eurovision 4 Congratulations to Austria, winner of the Eurovision Song Contest 2014!						
2	[2014-09-19	14:10:47]	===	#tinkerforge 3						
3	[2014-09-19	14:10:47]	===	##new 3 Welcome to ##NEW						
4										

这是第一轮文本分列的拆分结果。

(2) 我们还需要执行一次文本分列，但这次只需要操作D列就可以了，并且这回使用分割字符功能来替代固定列宽。首先让我们对数据做一下分析，频道名称#eurovision和用户数（4）之间有两个空格，而在4和频道描述之间也有两个空格。就算采用了文本分列方式，我们也无法使用两个空格作为分隔符号（因为这个功能只允许我们使用单个字符），所以我们需要使用查找和替换功能来把两个空格替换成一个从未使用过的字符。（先使用查找功能来确定我们选择的分隔符号确实没有使用过。）这里我选择了符号^。

这一步看起来不太优雅，如果你有什么更好的方法，尽管放手做吧。与其他一些工具相比，Excel的查找和替换功能是非常有限的。稍后我们会在3.2节学习正则表达式的使用方法。

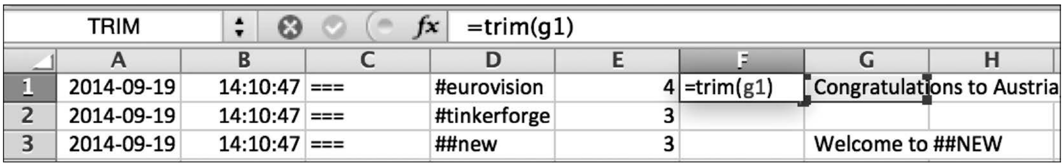


添加一个不太常见的分隔字符有利于我们拆分剩余的字段。

(3) 现在可以使用查找和替换删除A列和B列中的字符[和]，把它们替换成空。（请在开始查找和替换之前先选中这两列，以免意外删除工作表里其他列中所含有的相同字符。）

遗憾的是，有时Excel做得太多了，它会把日期格式转化成我们不太想要的：9/19/2014。如果你想让这些日期恢复到以往的样子（2014-09-19），请选择这个列，然后使用设置单元格格式将日期格式指定为yyyy-mm-dd。

(4) 在列F中，字符串的开端还有额外的空格字符。使用`trim()`函数可以去除字符串前面的空格字符。在列F旁边插入一个新列，然后调用`trim()`函数，请看下面的截图。



这就是使用`trim()`函数去除字符串首尾空格字符之后的效果。

(5) 我还可以使用`clean()`函数来完成文本剪裁。这个函数会去除ASCII表中前32个字符：就是那些可能混杂在频道描述字段中的所有非打印控制字符。此外，还可以把`clean()`套在`trim()`外面使用：`clean(trim(g1))`。

(6) 选中单元格F1，按住右下角并向下拖拽，这样F列其余的单元格也都会应用`clean(trim())`。

(7) 选中F列，复制，再用选择性粘贴粘贴数值，现在我们可以删除G列了。

(8) 删掉G列。好了，数据清洗完毕。

3.1.2 字符串拆分

在Google Spreadsheets中有一个轻量级的文本分列功能，`split()`函数，但Excel没有提供这个函数。这个函数的作用就是接收一个字符串值然后把它分割成几段。而你需要提供足够多的新列来容纳分割后的数据。在接下来的演示中，我们还要使用前例中的数据，在其基础之上创建几个新列来放置来自D列分割之后的数据。

fx =split(D1,"-")

	A	B	C	D
1	2014	9	19	2014-09-19
2	2014	9	19	2014-09-19
3	2014	9	19	2014-09-19

3.1.3 字符串拼接

函数`concatenate()`可以接受多个字符串参数（既可以是单元格引用也可以是带引号的字符串），然后把它们连接到一块儿放到新的单元格中。在下面的例子中，我们使用`concatenate()`函数把日期和时间两部分字符串组合到一起。这个函数在Excel和Google Spreadsheets中都可以使用，请看下面的截图。

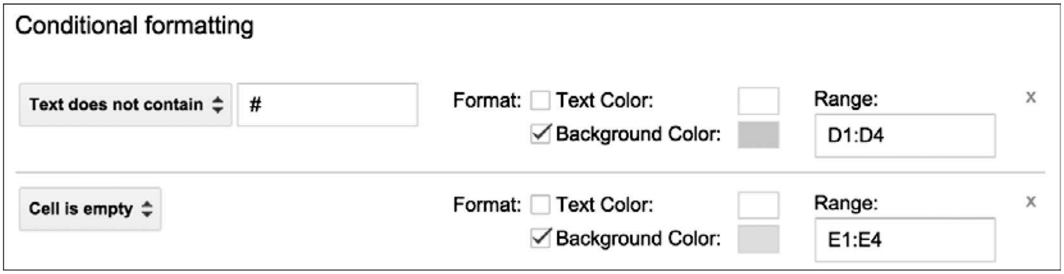
fx =CONCATENATE(B1," ",C1)

	A	B	C
1	2014-09-19 14:10:47	2014-09-19	14:10:47
2	2014-09-19 14:10:47	2014-09-19	14:10:47
3	2014-09-19 14:10:47	2014-09-19	14:10:47

1. 使用条件格式化查找异常数据

Excel和Google Spreadsheets都提供了条件格式化功能。条件格式化使用一组规则来改变满足标准（条件）的单元格外观（格式）。因此，我们可以利用这个功能来找出数据中的高值、低值、遗漏的值或是不正确的值。只要我们找到这些异常数据，就可以对它们展开清洗操作了。

下面的例子向我们演示了如何使用Google Spreadsheets的条件格式化功能，在样本数据中找出频道名称里没有包含符号#的记录以及聊天人数为空的记录。



下面是执行结果，被定位到的单元格背景色都发生了变化，这些单元格的内容要么是没有以符号#开头的D列数据，要么是含有空值的E列数据。现在很容易找到这些有问题的值。

	A	B	C	D	E	F
1	2014-09-19 14:10:47	2014-09-19	14:10:47	#eurovision	4	Congratulations to Au
2	2014-09-19 14:10:47	2014-09-19	14:10:47	#tinkerforge	3	
3	2014-09-19 14:10:47	2014-09-19	14:10:47	##new	3	Welcome to ##NEW
4	2014-09-20 14:10:48	2014-09-20	14:10:48	/END OF LIST		

2. 使用排序查找异常数据

如果待检查的数据过多，我们就需要考虑使用排序功能来查找问题数据了。不管是在Google Spreadsheets还是Excel中，我们都可以选中要排序的列，使用数据菜单上面的排序功能来对数据进行排序操作。对于大部分字段来说，排序实现起来都很容易，特别是查找像单元格D4那样的数据的时候。

但遇到E列这种缺少数据的情况，排序还能派上用场吗？或许只要把缺少数据的记录集中起来就可以把它们删除了。单元格E4中的值是空的。还记得第2章中所提到的NULL（Google Spreadsheets把它当作空处理）吧，这种空值是不能与任何值进行比较的，因此，不论对E列采用升序排列还是降序排列，排序之后它们总是停留在数据列表底部。

3. 往MySQL中导入电子表格数据

现在电子表格中的数据已经清洗完毕，接下来就要把数据放到数据库长久保存下来。

(1) 从电子表格创建CSV数据

许多数据库系统都提供从CSV文件导入数据的功能。如果你使用的是MySQL，可以使用一个叫作`LOAD DATA IN FILE`的命令把数据从分隔文件加载到数据库中，至于分隔符号，你完全可

以自己指定。首先让我们看一个命令的使用示例，然后再根据所需参数从Excel中创建数据文件。

在MySQL的命令行下运行下面的命令：

```
load data local infile 'myFile.csv'
  into table freenode_topics
  fields terminated by ','
  (dateOfTopic, channel, numUsers, message);
```

命令运行的前提是数据库中的表已经创建完毕。在这个例子中，表的名字是freenode_topics，其中指定了四个字段，它们位于SQL命令中的最后一行。

其中使用到的CSV文件名字为myFile.csv，里面包含的字段顺序应该与表中的字段顺序一一对应，分隔符号为逗号。

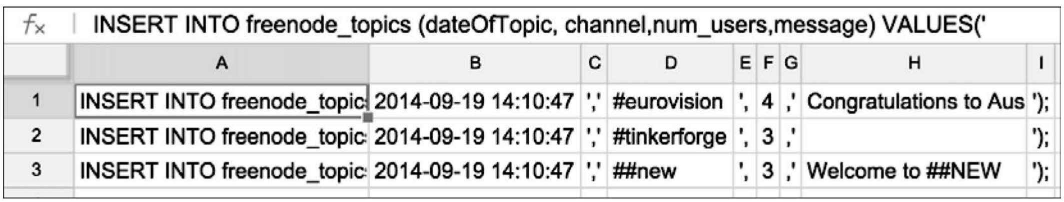
在Excel中，CSV文件的创建方法为，从文件菜单中选择另存为，然后从保存类型的列表中选择CSV（逗号分隔）。Google Spreadsheets的创建方法与之类似，File | Downloads | CSV。不管使用哪种工具，只要把文件保存到本地系统，就可以马上启动MySQL客户端，运行前面的命令行导入数据。

> 如果你不喜欢使用MySQL命令行客户端的话，可以选择MySQL的Workbench图形化客户端或是PhpMyAdmin这样的工具，把CSV上传到数据库服务器。在使用PhpMyAdmin的时候需要注意，它对上传的文件大小是有限制的（目前是2 MB）。

(2) 使用电子表格生成SQL

如果你不能把前面讨论的CSV数据文件加载到数据库——不管出于什么原因，也许是因为权限配置有误，或是文件大小受限，这时可以考虑使用另一种方法把数据导入数据库。这个方法乍一看起来会让人觉得有点奇怪，但它确实会节省不少时间。这个方法就是在电子表格应用内部构造INSERT语句，然后在数据库中运行生成的命令。

如果电子表格中的每一列都代表着数据库中的一个字段，我们只需围绕各个字段添加SQL命令INSERT所需的组件（引用字符串、圆括号、命令、分号），然后将结果拼接起来形成最终完整的INSERT命令即可。



fx INSERT INTO freenode_topics (dateOfTopic, channel,num_users,message) VALUES('

	A	B	C	D	E	F	G	H	I
1	INSERT INTO freenode_topic	2014-09-19 14:10:47	','	#eurovision	',	4	,'	Congratulations to Aus	');
2	INSERT INTO freenode_topic	2014-09-19 14:10:47	','	#tinkerforge	',	3	,'		');
3	INSERT INTO freenode_topic	2014-09-19 14:10:47	','	##new	',	3	,'	Welcome to ##NEW	');

使用函数`concatenate(A1:I1)`将列A:I中所有内容连接之后会得到下面的`INSERT`语句：

```
INSERT INTO freenode_topics (dateOfTopic, channel, num_users,
  message) VALUES('2014-09-19 14:10:47', '#eurovision', 4,
  'Congratulations to Austria, winner of the Eurovision Song Contest
  2014!');
```

我们可以把生成的命令粘贴到界面友好的前端应用程序中去，如PhpMyAdmin或MySQL Workbench。或者是（使用文本编辑器）把这些命令存为文本文件，每条`INSERT`语句彼此之间紧密相连。我创建的文件名字为inserts.sql。现在该使用命令行和MySQL客户端导入数据文件了，请参考下面的例子：

```
$mysql -uusername -p -hhostname databasename < inserts.sql
```

或者是利用MySQL命令行客户端里的`source`命令导入文件：

```
$mysql -uusername -p -hhostname
[enter your password]
> use databasename;
> source inserts.sql
```

这两种操作都可以把数据插入到MySQL数据库中。如果脚本不那么大，你也可以使用MySQL Workbench这样的图形化客户端导入数据。但在加载较大的脚本时则需要多加小心，因为客户机在加载几百G的数据量时很可能会内存不足。我个人比较喜欢第二种方法（`source`），因为它会在成功插入数据之后打印出成功执行的消息，这样我就会知道命令是没有问题的。

如果你还不太清楚怎么创建inserts.sql文件的话，那么下一节的内容你可得好好读读。我们要讲述的文本编辑器知识绝对超乎你的想象！

3.2　文本编辑器里的数据清洗

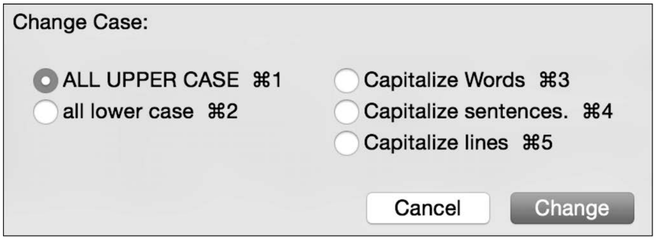
我们已经在第2章中了解到，文本编辑器是读写文本文件的首选方案。这听起来挺有道理的，而且再正常不过了。其实我不是要老生常谈，非得告诉你哪个文本编辑器有什么样的超酷功能，或者是教你如何用文本编辑器帮助整天需要与文本文件打交道的程序员和数据清洗人员完成他们的工作。我只是想向大家介绍文本编辑器中最为常用的一些功能而已。

> 实际上，在每个操作系统中都有一大把文本编辑器可用。有的需要付费，但也有许多免费的。在这一章中，我用的是Text Wrangler，它是OS X上的一款免费编辑器（产品地址为：http://www.barebones.com/products/textwrangler）。在这一章中介绍的许多功能在其他编辑器中也都能找到，比如Sublime，但你最好还是先查查这些编辑器提供的文档，因为不是所有功能或工具的位置都那么明显。

3.2.1　文本调整

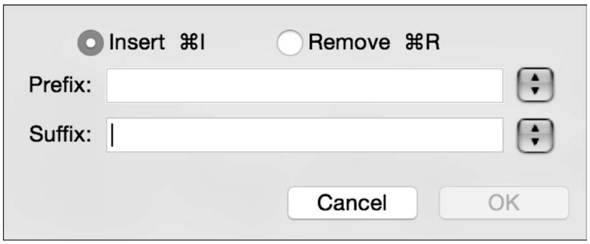
我们选用的文本编辑器内置了许多用于文本操作的功能。这里将要描述的大部分功能都与数据清洗任务有关。请记住，在数据清洗的过程中你可能需要针对同一个数据文件多次运行不同的程序，在这种情况下，我们可以用第1章中学到的技巧来清楚地传达数据清洗的工作内容。

改变大小写是数据清洗工作中很常见的任务。很多时候，我们拿到的数据要么都是小写，要么都是大写。而利用下面截图中显示的Text Wrangler对话框，我们可以改变选中文本的大小写。其中每个功能选项的键盘快捷键都显示在最右边。



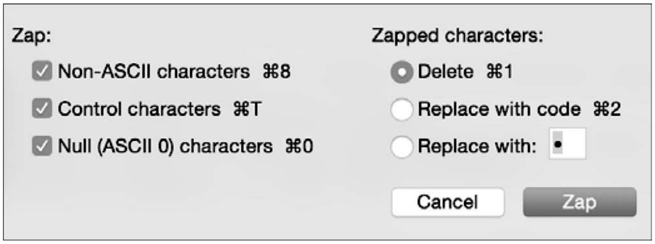
这些功能选项包括把文本全部变成大写、全部变成小写，以及把每个单词、句子或每一行文本的首字母变成大写。

另外还有一种常见的任务，就是在选中的文本上为其每一行添加或删除前后缀。可能在某一天构建文本分类器的时候我就需要在大段文本上应用这项操作。比如为每一行文本的末端追加逗号与该行的类别划分所对应的类别名称（肯定或否定该行的类别所属）。下面显示的是Text Wrangler前后缀操作对话框。需要注意的是，你可以使用插入功能或者是删除功能，但没法在一次操作中同时使用这两种功能。如果你需要使用这两种功能的话，那就先执行其中的一个，然后再执行另一个。



Zap Gremlins（一种将非ASCII字符转换为其他字符的方法）对文本编辑器来说绝对是一项锦上添花的功能。Windows版本的TextWrangler和Sublime编辑器都有这个功能。在`zap gremlins`的过程中，文本编辑器会查找那些不在常规字符集范围之内的所有字符，比如控制字符、NULL字符和非ASCII字符。我们可以选择删除这些字符或是使用这些字符所对应的编码来替代它们。或者干脆用我们指定的字符来作为替代字符。这样做的好处就是在后续的处理中更容易找到这些

字符曾经出现过的地方。



3.2.2　列选模式

当文本编辑器处于列选模式的时候，这意味着你不仅可以按行进行文本选取，还可以按列选取。下面是一个正常选取模式（非列选模式）下的例子：

```
[2014-09-19 14:10:47] === #eurovision   4   Congratulations to Austria,
[2014-09-19 14:10:47] === #tinkerforge   3
[2014-09-19 14:10:47] === ##new   3   Welcome to ##NEW
[2014-09-19 14:10:47] === #postcyberpunk   3   Postcyberpunk related ma
[2014-09-19 14:10:47] === #osi   7   The Open Source Initiative
[2014-09-19 14:10:48] === #apg-dev   3   APG - latest version: 1.1.1 -
```

下面是列选模式下的文本选取例子。按住Option键可以按列选取文本。一旦文本被选中，你就可以对它们进行删除操作，剪切或粘贴到剪切板，或者是像上一节讨论的那样调整。

```
[2014-09-19 14:10:47] === #eurovision   4   Congratulations to Austria,
[2014-09-19 14:10:47] === #tinkerforge   3
[2014-09-19 14:10:47] === ##new   3   Welcome to ##NEW
[2014-09-19 14:10:47] === #postcyberpunk   3   Postcyberpunk related ma
[2014-09-19 14:10:47] === #osi   7   The Open Source Initiative
[2014-09-19 14:10:48] === #apg-dev   3   APG - latest version: 1.1.1 -
```

不过这个功能有一些限制。

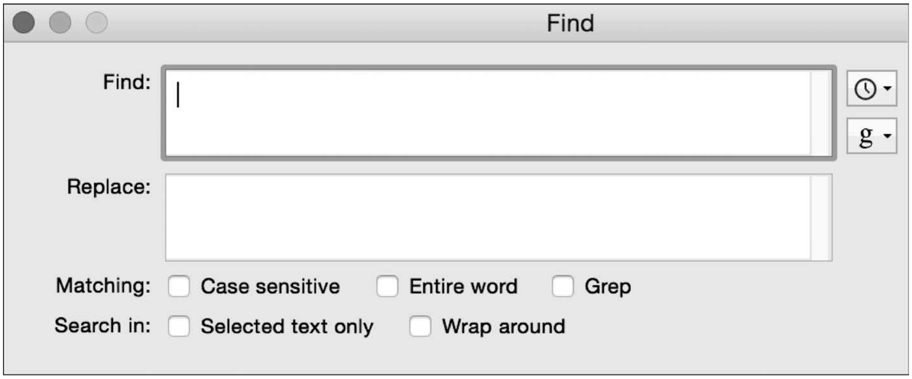
- 因为每个字符都独占一列，所以显示字符时使用非比例的打字机字体会比较好一些。
- 在Text Wrangler中，列选模式只有在自动换行模式关闭的情况下才能使用。关闭自动换行意味着文本将一直向右延展，不会折行。
- 在Text Wrangler中，要绘制的列的垂直高度，必须能够手动完成，因此这一技术只适用于少量数据（几百行或几千行，不太可能是成千上万行）。

3.2.3　加强版的查找与替换功能

不得不说，文本编辑器在文本操作这方面真的很出色。但它的列选模式总是让人感觉有些怪

怪的，因为这看起来更像是电子表格应该干的事情。同样，如果你了解了文本编辑器的查找与替换功能，电子表格的查找与替换也会显得拙劣不堪。

下面显示的是Text Wrangler中的查找对话框窗口截图。里面提供了区分大小写的查找，折行查找，在选中的文本中查找，以及单词全字符匹配或单词部分字符匹配的查找。你可以往文本框里粘贴特殊字符、空白字符（包括制表符和换行符）、表情符号等。查找对话框右侧的下拉按钮中还额外提供了一些功能选项。顶部带有时钟图标的按钮会保留最近一段时间内所使用过的一些查找与替换操作记录。底部带有字母g的按钮包含了一个内置的常用搜索模式列表，我们可以通过列表最下方的选项添加自定义模式。



查找与替换中最强大的功能之一就是Grep选项。勾选上这个选项之后，我们就可以使用正则表达式来进行查找。简单地讲，正则表达式（regex）是一种以特殊语言编写的匹配模式，由各种符号组成，其设计目的就是为了匹配字符串文本。关于正则表达式的讲解超出了本书的讨论范围，我们只要明白它们用处很大，在数据清洗的时候可能需要时不时地用到，这就足够了。

> Text Wrangler界面中使用Grep作为复选框的名字——而不是RegEx或Match——是因为有好几种不同的正则表达式模式匹配语法。而Text Wrangler给我们的提示信息是，它所使用的语法来自于Grep——一种最初为Unix编写的程序，它的语法也是广泛使用的正则表达式语法之一。

接下来让我们看看数据清洗中常用的正则表达式符号。如果你想学习更为复杂的模式，最好还是查阅一些专门的书籍或是网站，其中列举了各种稀奇古怪的正则表达式语法。

符　号	使用说明
$	匹配一行的结尾位置
^	匹配一行的开始位置
+	匹配一个或多个指定的字符
*	匹配0个或多个指定的字符

（续）

符　号	使用说明
\w	匹配0~9或A~z中任意字符。如果不想匹配这些字符，可以使用\W
\s	匹配空白字符（制表符、换行符或回车符）。如果不想匹配这些字符，可以使用\S
\t	匹配制表符
\r	匹配回车符。如果要匹配换行符，可以使用\n
\	转义字符。用于取消紧随其后的字符在正则表达式中的特殊含义

下面是一些查找与替换的组合应用实例，有助于我们在正则表达式中学习Text Wrangler。记得把复选框Grep勾选上。如果替换一栏里面什么都没有，这表示对话框中的Replace文本框需要留空。

查找	替换	使用说明
\r		找出换行符（行终止符）并把它替换成空。换种说法就是“把多行合并成一行”
^\w+	-	匹配一行的开始位置，并要求后面至少有一个字符。成功匹配的内容会被-符号替代
\\r$	\[end\]	查找所有以\r（反斜杠后面是字母r）为结束字符的行，然后用字符串[end]来取代。需要注意的是[和]在正则表达式中有着特殊的意义，所以在使用的时候需要对它们进行转义

如果你对怎么编写正则表达式有些发愁的话，大可不必。首先回想一下3.2.1节中提到的，包括Text Wrangler在内的大多数文本编辑器都内置了许多常用的正则表达式，这足以满足一般性的查找与替换操作。所以，你可能会发现你并不需要经常自己编写正则表达式。其次，正是因为正则表达式的强大功能与实用性，无论你什么时候需要它，总是有很多的在线资源可供利用，凭借这些资料你完全可以写出复杂的正则表达式样式。

我个人比较喜欢使用的正则表达式资源是Stack Overflow（http://stackoverflow.com）和regular-expresions.info（http://regular-expressions.info）。另外，通过搜索引擎我们还可以找到许多正则表达式的测试网站。这些网站可以让你对编写出来的正则表达式进行测试。

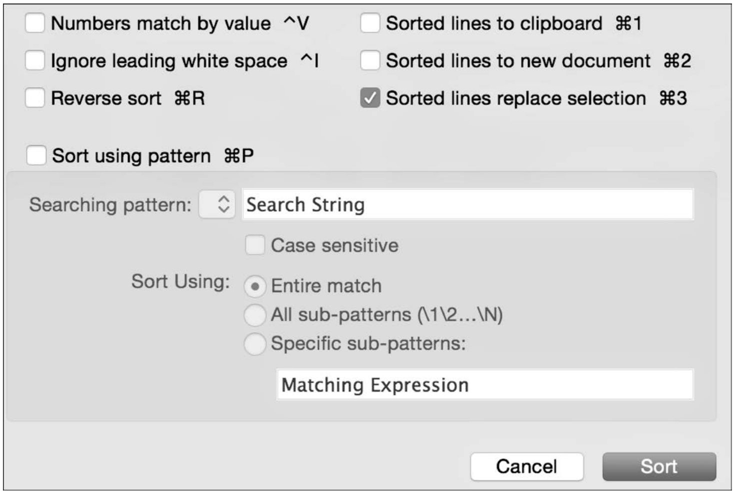
箴言警句

请小心使用在线的正则表达式测试工具，因为这些工具常常是教授某一种正则表达式语法，比如适用于JavaScript、PHP或Python的语法等。当中有的正则表达式行为可能会和文本编辑器里的保持一致，而有的表现则可能不同。根据操作的复杂程度不同，最好还是先创建一份文本数据副本（或是抽取一小部分样本到一个新的文件中），然后在文本编辑器中试验正则表达式语法。

3.2.4　文本排序与去重处理

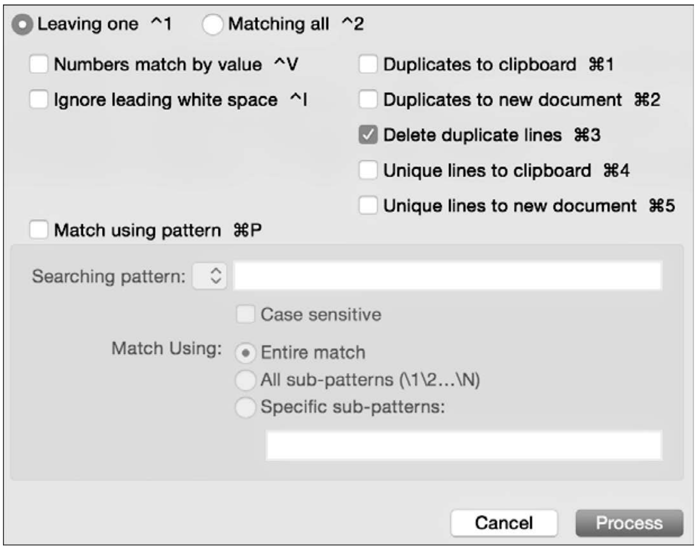
在初试正则表达式的功能之后，我们会注意到文本编辑器有时在别的功能菜单下也会有模式匹配，比如数据排序和去除重复处理。请看下面的排序对话框，通过它我们更好地理解正则表达

式是如何应用在多行排序操作上的：



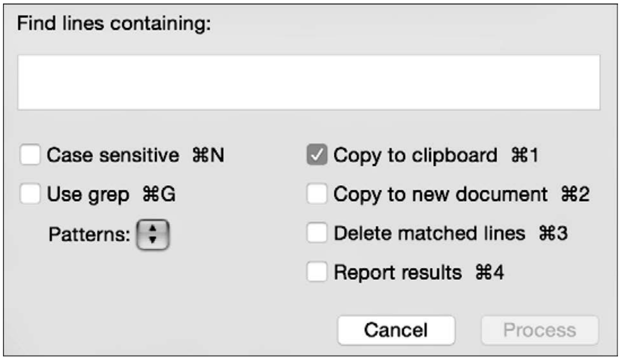
在这个例子当中，我们需要勾选Sort using pattern并输入正则表达式模式来进行排序操作。去重处理的对话框也差不多。你可以向编辑器指明是需要保留原始行还是直接删除它。更有意思的是，你还可以把重复的内容移到另一个文件或剪切板里去，如此一来你就可以在别的地方使用它们了，比如跟踪被删除数据什么的。

> 在做数据清洗的时候，最好还是把删除的行保存在单独的文件中，以防万一需要。



3.2.5 Process Lines Containing

Text Wrangler还有一个方便的功能叫作Process Lines Containing。这个功能把查找功能（支持正则表达式的使用）和逐行处理整合到一起了，这样我们就可以把删除的行转移到另一个文件或剪切板里，或是删除满足匹配条件的行。



3.3 示例项目

在这个示例项目中，我们将要下载一个电子表格，并使用Excel或是文本编辑器来对它进行数据清洗，之后再进行简单的数据分析。

3.3.1 第一步：问题陈述

这个项目受到了美国高等教育编年史所提供的数据的启发，高等教育编年史是一个记录各大院校新闻事件的出版物。2012年，他们推出了一个被称作“哪些学校与你的大学互为同级别院校”的交互功能。在这个功能中，用户可以往表单中输入美国各大院校的名字，然后就能看到一份带有交互功能的可视化报告，报告中会列出有哪些学校把你输入的目标学校当成它们的同级别院校。（同级别院校指的是在某些方面水平相当的一些院校。）原始数据来自美国政府的报告，而高等教育编年史对数据做了可视化处理并对公众开放。

在这个例子中，我们的附加问题是：当X大学出现在列表名单中时，还会有哪些大学也在这个列表上呢？要回答这个问题，我们得先找出所有与X大学同时出现在数据列表中的大学，然后统计它们名字出现的次数。

3.3.2 第二步：数据收集

在这个步骤中，我们需要先完成数据收集的过程，然后才可以进行逐步清洗。所以，接下来

要谈的就是采用哪些操作完成项目数据的收集。

1. 下载数据

项目中使用到的数据可以从http://chronicle.com/article/Who-Does-Your-College-Think/134222/原文中下载，或者是直接访问https://s3.amazonaws.com/03peers/selected_peers.csv.zip下载。

这个文件是经过压缩处理的，你得准备好合适的解压工具才能解压。

2. 了解数据

ZIP文件夹里的文件扩展名为.csv，它就是一个CSV文件，共有1686行数据，其中包括标题行。逗号分隔符把数据拆分为两个部分：第一列是大学的名字，第二列是大学列表，这些大学被当作目标大学的同级别院校。这些同级别院校之间采用管道字符（|）进行分隔。请看下面的例子。

```
Harvard University,Yale University|Princeton University|Stanford University
```

示例中第一列表示哈佛大学是目标大学，第二列表示耶鲁、普林斯顿、斯坦福被列为哈佛大学的同级别院校。

3.3.3　第三步：数据清洗

这个示例的目标是找出一个特定的大学和同时列在名单上的其他同级别院校，所以我们首先要做的就是去除那些没有包含目标大学的数据。处理完成之后我们可以从数据文件中得到一份包含各个学校的清单列表。走到这一步时，数据应该都已经清洗完毕，然后要做的就是数据分析与可视化处理了。

1. 抽取相关的数据行

先让我们比较两种数据抽取的方法：第一种是使用电子表格应用程序，另一种则是使用这章介绍过的文本编辑器技术。

(1) 使用电子表格

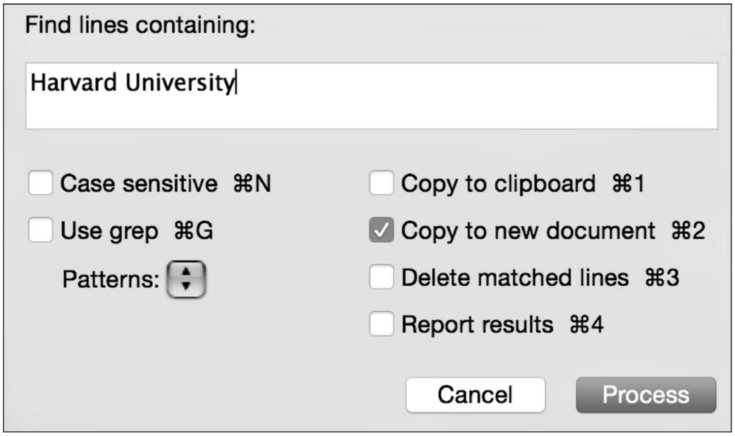
用电子表格应用程序打开文件，然后使用条件格式化来高亮显示包含哈佛大学（或是你选择的其他学校）的行记录，然后删除没有高亮显示的记录。

(2) 使用文本编辑器

用文本编辑器打开文件，然后使用Process Lines Containing找出包含Harvard University（或是你选择的其他学校）的行记录，然后把它们复制到新的文件中。

无论使用哪种方法，最终的处理结果都应是一个包含26行记录的文件，每条记录都包含

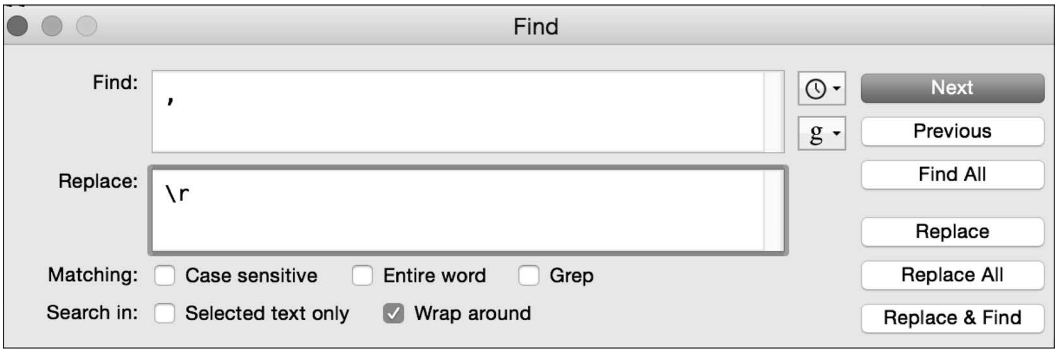
Harvard University。



2. 行数据转换

文件里已经有26条记录了，每条记录都列举了好几所大学（数据很宽）。我们希望能用Python读出文件中的数据并在适当的时机统计出这些大学出现的频率。因此，我们要把这些数据调整一下，每行只放一个大学。

想要让每行只包含一所大学，我们得使用文本编辑器并连续执行三次查找与替换操作。首先要做的是找出逗号并把它们替换成/r（回车符）。



接下来找出管道符号（|），把它们也替换成/r（回车符）。到这里为止，文件中一共包含749行数据。

最后要做的是删除Harvard University。（请记住，我们只需关注与哈佛大学为同级别的院校，所以在统计次数的时候不需要考虑哈佛大学。）在Text Wrangler中，我们可以把字符串Harvard University\r放到文本框Find中，并让文本框Replace留空。这次操作会删除26行记录，现在总行数应该是723。

3.3.4　第四步：数据分析

由于我们的侧重点是数据清洗，所以并不会在数据分析和可视化上面花费过多的时间，在这里我们只需要使用下面的Python脚本，找出那些与哈佛大学为同级的院校在列表中出现的次数：

```
from collections import Counter
with open('universities.txt') as infile:
    counted = Counter(filter(None,[line.strip() for line in infile]))
    sorted = counted.most_common()
    for key,value in sorted:
        print key, ",", value
```

3

结果一共显示了232所大学的名称，以及这些大学被列举的次数。下面是部分结果数据。我们对此的理解应为：

当哈佛大学被列为同级别院校时，耶鲁大学一共被列举26次：

```
Yale University , 26
Princeton University , 25
Cornell University , 24
Stanford University , 23
Columbia University in the City of New York , 22
Brown University , 21
University of Pennsylvania , 21
```

走到这一步，你完全可以凭着手上的这份列表清单（或是只截取列表清单中的一部分），把它制成条形图、文字云，或者是应用在其他更有说服力、更让人心动的可视化实现上。结果数据本身是用逗号分隔的，所以你可以直接把它们粘贴到电子表格中做进一步的分析。但就这个项目而言，我们已经回答了最初提出的问题，找到与给定目标大学互为同级别院校并且被列举次数最多的大学。

3.4　小结

在这一章中我们学会了一些非常实用的数据清洗技巧，并使用了两个工具：文本编辑器和电子表格应用程序。我们先学习了电子表格中的各种函数，如用于拆分数据、移动、查找与替换、格式化以及整合数据。接着学习了怎么使用文本编辑器，包括许多内置功能，以及如何高效利用查找与替换功能及正则表达式。

在接下来的一章中，我们会把到目前为止所学的多种技术结合起来，去完成一些难度更大的文件转换。当中有许多技术都是基于前面两章所学的知识，就像文本编辑、正则表达式、数据类型、文件格式等。准备好了吧，向数据清洗进发吧！

第 4 章

讲通用语言——数据转换

去年夏天，我报名参加了当地一所烹饪学校开设的奶酪制作课程。第一堂课的内容就是做乳清奶酪。当时令我兴奋的是，仅仅使用牛奶和脱脂奶就可以在一个小时内完成乳清奶酪的制作，而且脱脂奶可以由牛奶和柠檬汁加工制成。在厨房里，食物的原料可以转变成其他原料，进而成为餐桌上的美食。而在数据科学厨房中，我们也常常会把数据从一种格式转换成另一种格式。有时我们会把多个数据集合并到一起，有时会改变数据集的存储方式，这时就需要进行数据转换，以便进行各种数据分析。

所谓*通用语言*，就是讲不同语言的人在对话中采用的一种公共标准。而在数据转换的时候，也有几种可以作为公共标准的数据类型。在第2章就曾提到过一些。JSON和CSV是其中最为常见的两种数据标准。在这一章中，我们将要学习以下内容。

- 如何利用软件工具或语言（Excel、Google Spreadsheets和phpMyAdmin）把数据快速地转换为JSON和CSV格式。
- 如何编写Python和PHP程序来生成不同的文本格式，并在这些文本格式之间实现互相转换。
- 如何利用数据转换来完成实际项目任务。在这一章的项目中，我们将使用netvizz从Facebook上下载朋友圈数据，随后对这些数据进行清洗并把它们转换成JSON格式，这样就可以在D3中做出可视化的社交网络图。之后我们将换一种方案重新再做一次数据清洗，并把数据转换成Pajek格式，这种格式可以直接应用在一种名为networkx的社交网络包中。

4.1 基于工具的快速转换

针对少量或中量数据的转换，最快最容易的方法就是直接使用一切可以加以利用的软件和工具。有时候，我们使用的应用软件很可能已经包含了数据转换功能，利用这些功能我们可以轻松地得到需要的数据格式。就像在第3章中提到的技巧和窍门一样，只要有可能，还是应该好好利用工具中所提供的各种隐藏的功能。不过一旦遇到数据量过大，或是超出应用程序转换能力之外的情况，就得使用4.2节和4.3节中介绍的编程方案来解决。

4.1.1 从电子表格到 CSV

把电子表格数据保存为分隔文件非常容易实现。Excel和Google Spreadsheets都在文件菜单中提供了*另存为*的功能，在这个功能中，我们只需选择CSV格式即可。此外，Google Spreadsheets还提供把电子表格存为Excel文件和制表符分隔文件的功能。但在保存CSV文件的时候，有一些限制我们还是有必要了解一下的。

- 不论是使用Excel还是Google Spreadsheets，在使用*另存为*功能的时候，只有当前的工作表中的内容会被保存。这是因为CSV文件只能描述一组数据集；因此，它无法保存来自多个工作表的数据。如果你的电子表格中有多个工作表的话，你得分别把每一个工作表单独保存为CSV文件。
- 在使用这两种工具生成CSV文件的时候，基本上没有什么自定义选项，比如Excel只能以逗号作为分隔符（对于CSV文件来说，这点还说得过去）来保存文件，而且在双引号封闭数据和换行符方面也不提供定制功能。

4.1.2 从电子表格到 JSON

与CSV格式相比，JSON格式的转换要稍微复杂一些。虽然Excel没有提供直接转换JSON格式的功能，但有一些在线转换工具倒是声称它们可以实现JSON格式的转换。

而在Google Spreadsheets里，提供了一个通过URL就能访问的JSON转换器。不过这个方法也有一些缺陷，首当其冲的一条就是你必须通过Google Spreadsheets在互联网上发布你的文档（哪怕是临时性的），因为只有这样才能访问它的JSON版本数据。此外，你还需要使用由长长的数字修饰的URL，这样做的目的是为了可以在互联网中定位你的电子表格文件。而最终产生的JSON数据中会含有大量的信息，其中一部分可能是你压根儿就不想要的或是不需要的。即便如此，还是让我们先看看怎么通过一步步操作把Google电子表格文件转换成JSON格式的文件。

1. 第一步：把Google电子表格发布到互联网

当你创建并保存好一份Google电子表格文件之后，从File菜单中选择Publish to the Web。然后连续点击通过后面出现的对话框（我全部使用默认的选项）。全部完成之后，你就可以通过URL来访问JSON格式的数据了。

2. 第二步：创建正确的URL

通过下面的URL样式，我们可以从已经发布出来的Google电子表格中创建JSON数据：

http://spreadsheets.google.com/feeds/list/key/sheet/public/basic?alt=json

在这个URL中，你需要根据具体的电子表格对其中三项内容进行调整。

- list：（可选）如果你想让每个单元格通过它们所对应的引用（A1、A2等）独立地显示出来，可以把list修改成单元格（cell）。如果你想让每一行数据作为一个完整的实体存在，那就请保留URL中的list。
- key：把URL中的key改换成文件在Google内部所对应的长长的唯一编码。在电子表格的URL中，就像你在浏览器中所看到的那样，这个key是位于两个斜线之间的，恰好在/spreadsheets/d之后，请看下面的截图。



- sheet：如果把上面示例中URL里包含的单词sheet修改为od6，这表明你只想转换第一个工作表。

> od6是什么意思呢？Google使用编码来表示每一个工作表。但是编码并非以严格的数字顺序进行排列。关于编码方案的讨论请参见Stack Overflow上的讨论，其中包括了问题和相关答案，地址为http://stackoverflow.com/questions/11290337/。

要验证这个过程，我们需要创建一个Google电子表格来保存第3章中最后一个项目里生成的各个大学信息以及相关的统计数据。前三行数据内容如下：

Yale University	26
Princeton University	25
Cornell University	24

而我用来访问JSON数据的URL如下：

http://spreadsheets.google.com/feeds/list/1mWIAk_5KNoQHr4vFgPHdm7GX8Vh22WjgAUYYHUyXSNM/od6/public/basic?alt=json

把这段URL粘贴到浏览器之后，我们就可以得到一份JSON格式的数据。结果共有231条记录，每一条记录的样式都与下面的代码片段类似。为了方便阅读，我把这条记录进行了格式化并添加了一些换行符：

```
{
  "id":{
    "$t":"https://spreadsheets.google.com/feeds/list/1mWIAk_5KN
    oQHr4vFgPHdm7GX8Vh22WjgAUYYHUyXSNM/od6/public/basic/cokwr"
  },
  "updated":{"$t":"2014-12-17T20:02:57.196Z"},
  "category":[{
    "scheme":"http://schemas.google.com/spreadsheets/2006",
    "term" :"http://schemas.google.com/spreadsheets/2006#list"
```

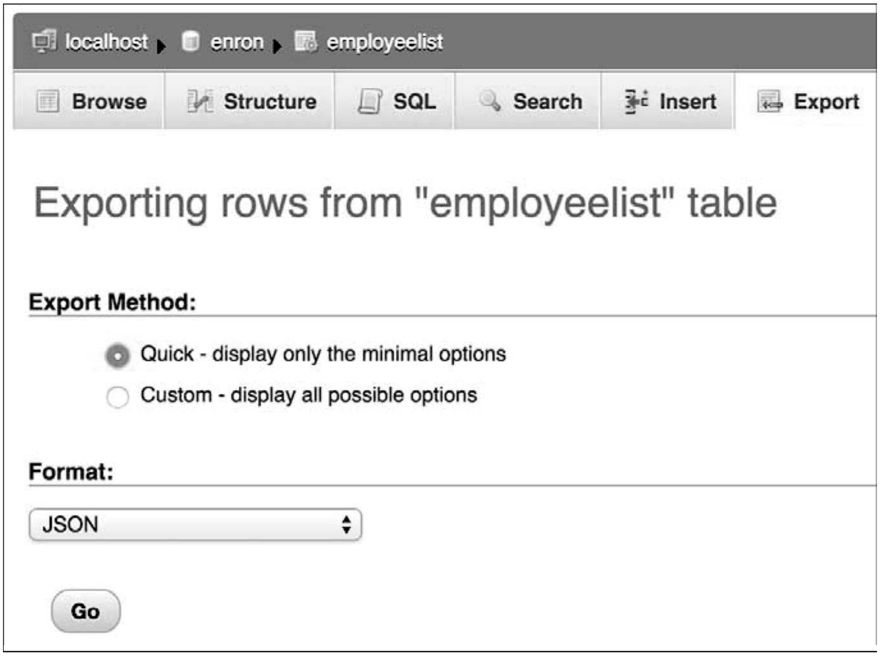
```
    }],
    "title":{
      "type":"text",
      "$t" :"Yale University "
    },
    "content":{
      "type":"text",
      "$t" :"_cokwr: 26"
    },
    "link": [{
      "rel" :"self",
      "type":"application/atom+xml",
      "href":"https://spreadsheets.google.com/feeds/list/1mWIAk_5KN
      oQHr4vFgPHdm7GX8Vh22WjgAUYYHUyXSNM/od6/public/basic/cokwr"
    }]
  }
```


即便经过了重新格式化，JSON还是比较长，其中有很多信息不是我们想要关注的。但不管怎么说，我们已经成功地生成了功能齐全的JSON数据。假如我们需要使用程序来处理这个JSON，所有与电子表格相关的附加信息都可以忽略掉，只有标题和内容实体里的`$t`字段对应的值（如上例中的`Yale University`和`_cokwr: 26`）才是需要处理的。这些值在上面的JSON示例中很容易找到。如果你还想问是否有办法把电子表格转换成CSV数据，进而转换成JSON数据，答案是肯定的。我们将在4.2节和4.3节中进行详细说明。

4.1.3 使用 phpMyAdmin 从 SQL 语句中生成 CSV 或 JSON

在这一节中，我们要讨论如何在不借助任何编程方法的条件下，直接从MySQL数据库中导出JSON和CSV这两种格式的数据。

首先，phpMyAdmin是一种使用度较高的基于Web的MySQL数据库客户端程序。如果使用这个工具的高级版本，我们可以将一整张表的数据或是查询出来的结果数据直接输出为CSV或JSON格式的文件。这回我们还是使用第1章中提到的Enron数据库。从下面的截图中可以看出，在Export标签中可以选择JSON作为数据表employeelist的目标格式（也可以选择CSV作为目标格式）：



使用phpMyAdmin输出JSON格式的数据表

输出查询结果的做法与输出数据表很相似，只是不能使用屏幕上方的Export标签，而是在运行完SQL查询之后使用页面下方Query results operations里的Export，请见下图。



使用phpMyAdmin输出查询结果

为了测试一下输出功能，我们可以使用下面的查询语句来检索数据表employeelist：

```
SELECT concat(firstName, " ", lastName) as name, email_id
FROM employeelist
ORDER BY lastName;
```

在输出JSON格式的时候，从phpMyAdmin中我们会看到151条格式与下面类似的数据：

```
{
  "name": "Lysa Akin",
  "email_id": "lysa.akin@enron.com"
}
```

phpMyAdmin是一款很好用的工具，它可以高效地实现适量的数据转换，这些数据可以来自MySQL数据表，或是一条查询语句的结果。如果你使用的是别的RDBMS，它们很有可能也有一些专属的格式化选项，而这一切都得由你去探索发掘。

此外，还有一种可以完全撇弃phpMyAdmin的方法，只需使用MySQL命令行就能以我们自定义的格式输出CSV文件：

```
SELECT concat(firstName, " ", lastName) as name, email_id
INTO OUTFILE 'enronEmployees.csv'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM employeelist;
```

通过上面的语句我们可以得到指定名字（employees.csv）的逗号分隔文件，这个文件会在当前文件夹中生成。

那能不能以同样的方法生成JSON数据呢？很遗憾，这种方法是不支持JSON格式输出的。如果想要JSON格式，要么选择前面提到的phpMyAdmin解决方案，要么使用PHP或是Python来实现你自己的方案。以编程方式实现的解决方案会在接下来的几节中讨论，请继续阅读后面的内容。

4.2　使用 PHP 实现数据转换

在第2章中我们曾讨论过JSON的数字格式，当时我们通过一个PHP脚本，演示了连接数据库、运行查询、从查询结果构建PHP数组，并在最后把JSON格式的结果数据输出到屏幕上。在这一节里，我们先对前面例子的功能做一些扩展，直接把数据输出成CSV格式的文件而不是输出到屏幕上。然后再演示如何使用PHP读取JSON文件并把它转换成CSV文件，最后再演示反向转换操作。

4.2.1　使用 PHP 实现 SQL 到 JSON 的数据转换

在这一节中，我们要用PHP脚本连接enron数据库，然后运行SQL查询语句把查询结果输出为JSON格式的文件。那我们为什么要使用PHP脚本来替代phpMyAdmin呢？其中一个原因就是这种方案能够让我们在输出数据之前多做一些额外的处理，此外，如果我们预期的数据量超过了Web应用程序处理能力之外，也需要使用这种方案。

```
<?php
// 连接到数据库，设置查询语句，运行查询语句
$dbc = mysqli_connect('localhost','username','password','enron')
or die('Error connecting to database!' . mysqli_error());

$select_query = "SELECT concat(firstName, \" \", lastName) as
  name, email_id FROM employeelist ORDER BY lastName";

$select_result = mysqli_query($dbc, $select_query);
```

```
if (!$select_result)
  die ("SELECT failed! [$select_query]" . mysqli_error());

// ----JSON格式数据的输出----
// 构建一个适合输出json数据的新数组
$counts = array();
while($row = mysqli_fetch_array($select_result))
{
  // 将数据添加到json数组中
  array_push($counts, array('name'   => $row['name'],
  'email_id' => $row['email_id']));
}
// 将查询结果以json格式进行编码处理
$json_formatted = json_encode($counts);

// 写入json文件
file_put_contents("enronEmail.json", $json_formatted);
?>
```

这段代码中`file_put_contents()`用于指定最终输出的JSON文件位置。

4.2.2　使用 PHP 实现 SQL 到 CSV 的数据转换

下面的代码演示了如何使用PHP文件输出流从SQL查询结果中创建CSV文件。我们需要把下面的代码保存为.php文件之后放到网络服务器上，并确保文件所在的目录支持脚本运行，然后再通过浏览器请求这个文件。访问时CSV文件就会自动开始下载：

```
<?php
// 连接到数据库，设置查询语句，运行查询语句
$dbc = mysqli_connect('localhost','username','password','enron')
  or die('Error connecting to database!' . mysqli_error());

$select_query = "SELECT concat(firstName, \" \", lastName) as name, email_id FROM
  employeelist ORDER BY lastName";

$select_result = mysqli_query($dbc, $select_query);

if (!$select_result)
  die ("SELECT failed! [$select_query]" . mysqli_error());

// ----CSV格式数据的输出----
// 设置文件流
$file = fopen('php://output', 'w');
if ($file && $select_result)
{
  header('Content-Type: text/csv');
  header('Content-Disposition: attachment;
  filename="enronEmail.csv"');
  // 将每行结果数据都以csv格式写到文件中
  while($row = mysqli_fetch_assoc($select_result))
  {
```

```
    fputcsv($file, array_values($row));
  }
}
?>
```

输出文件的内容格式如下（这里只是截取前三行数据）：

```
"Lysa Akin",lysa.akin@enron.com
"Phillip Allen",k..allen@enron.com
"Harry Arora",harry.arora@enron.com
```

如果你想知道Phillip的电子邮箱中是否包含两个圆点符号，可以通过下面的查询语句快速地从Enron数据库中找出这样的数据：

```
SELECT CONCAT(firstName, " ", lastName) AS name, email_id
FROM employeelist
WHERE email_id LIKE "%..%"
ORDER BY name ASC;
```

结果显示一共有24条记录的电子邮件地址包含两个圆点符号。

4.2.3　使用 PHP 实现 JSON 到 CSV 的数据转换

下面的代码可以读取JSON文件并把它转换成CSV文件：

```
<?php
// 读取文件
$json = file_get_contents("outfile.json");
// 将JSON格式数据转换成关联数组
$array = json_decode ($json, true);
// 打开文件流
$file = fopen('php://output', 'w');
header('Content-Type: text/csv');
header('Content-Disposition: attachment;
filename="enronEmail.csv"');
// 循环数组数据，并把每行都内容都写入到文件中
foreach ($array as $line)
{
  fputcsv($file, $line);
}
?>
```

这段代码创建了一个与前面例子一样的CSV文件。这里需要注意函数`file_get_contents()`的用法，它会把文件的内容以字符串形式保存到内存中，如果你要查找的文件体积很大的话，建议考虑使用`fread()`、`fgets()`和`fclose()`等函数。

4.2.4　使用 PHP 实现 CSV 到 JSON 的数据转换

还有一种比较常见的任务是读取CSV文件并把其中的内容转换成JSON格式。在大多数情况

下，CSV的第一行数据保存的是标题信息。标题行会列出文件中每一个字段的名字，在构造JSON文件的时候，我们需要利用这些字段信息来充当JSON结构中的属性：

```
<?php
$file = fopen('enronEmail.csv', 'r');
$headers = fgetcsv($file, 0, ',');
$complete = array();

while ($row = fgetcsv($file, 0, ','))
{
  $complete[] = array_combine($headers, $row);
}
fclose($file);
$json_formatted = json_encode($complete);
file_put_contents('enronEmail.json',$json_formatted);
?>
```

代码输出的结果是基于前面创建的enronEmail.csv文件，JSON的属性取自标题行，最终的结果数据如下：

```
[{"name":"Lysa Akin","email_id":"lysa.akin@enron.com"},
{"name":"Phillip Allen","email_id":"k..allen@enron.com"},
{"name":"Harry Arora","email_id":"harry.arora@enron.com"}…]
```

实际的CSV文件中一共包含151条记录，这里只截取前三行。

4.3　使用 Python 实现数据转换

在这一节里，我们要讨论如何使用Python来完成CSV和JSON之间的转换。在列举的例子当中，我们将寻求各种不同的方式来完成目标任务，有可能是借助特殊的安装包，也有可能只使用纯粹的Python代码。

4.3.1　使用 Python 实现 CSV 到 JSON 的数据转换

如果使用Python的话，我们可以使用多种方法来把CSV数据转换成JSON格式。而让人最容易想到的就是使用内置的`csv`和`json`库。假设我们的CSV文件内容如下（这里只截取了前三行数据）：

```
name,email_id
"Lysa Akin",lysa.akin@enron.com
"Phillip Allen",k..allen@enron.com
"Harry Arora",harry.arora@enron.com
```

我们可以使用Python来写一个程序读取行信息并把它们转换成JSON格式：

```
import json
```

```
import csv

# 读取CSV文件
with open('enronEmail.csv') as file:
    file_csv = csv.DictReader(file)
    output = '['
    # 处理每一个目录
    for row in file_csv:
      # 两个实体之间加入逗号
      output += json.dumps(row) + ','
    output = output.rstrip(',') + ']'
# 把文件写入磁盘中去
f = open('enronEmailPy.json','w')
f.write(output)
f.close()
```

输出的JSON结果如下（这里只截取前两行记录）：

```
[{"email_id": "lysa.akin@enron.com", "name": "Lysa Akin"},
{"email_id": "k..allen@enron.com", "name": "Phillip Allen"},…]
```


使用这个方法的好处是简单直接，我们不需要安装额外的特殊库或是命令行工具，只需要基本的读（CSV）写（JSON）操作就能解决问题。

4.3.2 使用 csvkit 实现 CSV 到 JSON 的数据转换

第二种把CSV转换成JSON的方法需要依赖一个叫作csvkit的Python工具包。要想安装csvkit，我们需要使用Canopy，首先启动Canopy终端窗口（通过菜单Tools | Canopy Terminal），然后运行`pip install csvkit`命令。命令结束后，csvkit以及它要依赖的全部内容都会安装完毕。这个时候，你可以在Python程序中通过`import csvkit`访问csvkit，或者是像下面这样通过命令行访问：

```
csvjson enronEmail.csv > enronEmail.json
```

利用上面的命令和输入参数`enronEmail.csv`，我们可以快速、轻松地输出JSON版本的数据文件enronEmail.csvkit.json。

另外，csvkit包中还有一些其他有用的命令行程序，比如`csvcut`，它可以从CSV文件中提取任意指定字段中的数据，还有`csvformat`，它可以用于改变CSV文件中的分隔符或换行符号，或是与此相似的工作。`csvcut`程序的长处在于你可以根据自己的需要来定制想要处理的数据。还有一点要提的是，不论使用的是工具包中的哪个命令行程序，它们都支持以重定向的方式创建新文件。下面的命令行以`bigFile.csv`为参数，分别提取了第一列和第三列数据，并把结果保存到一个全新的CSV文件：

```
csvcut bigFile.csv -c 1,3 > firstThirdCols.csv
```

关于csvkit文档、下载与示例的更多信息，请访问http://csvkit.rtfd.org/。

4.3.3　使用 Python 实现 JSON 到 CSV 的数据转换

使用Python读取JSON文件并将它转换成CSV文件的过程非常简单：

```
import json
import csv

with open('enronEmailPy.json', 'r') as f:
    dicts = json.load(f)
out = open('enronEmailPy.csv', 'w')
writer = csv.DictWriter(out, dicts[0].keys())
writer.writeheader()
writer.writerows(dicts)
out.close()
```

这个程序利用一个名为enronEmailPy.json的JSON文件，输出一个对应的CSV版本结果文件enronEmailPy.csv，其中的标题行信息来自于JSON中的属性。

4.4　示例项目

在这一章中，我们集中精力研究了如何把数据从一种格式转换成另一种格式，这在数据清洗中是很常见的，常常在数据分析项目的其他工作之前进行。我们研究的重点是常见的文本格式（CSV和JSON）和常见的数据存储位置（文件和SQL数据库）。现在该是扩展数据转换基础知识的时候了，接下来我们将以一个实际的项目为出发点来看看怎么处理非标准化的情况——但还是针对文本数据格式。

在这个项目中，我们要开展一项关于Facebook社交网络的调研工作。因此，我们将要完成以下内容。

(1) 用netvizz下载Facebook的社交网络（好友以及好友之间的关系）并把这些数据保存为以文本为基础的地理数据格式（GDF）。

(2) 构建一个关于Facebook社交网络的图形，其中每一个人都用一个节点表现，人与人之间的关系使用连接线（也称为边）来表现。要想完成这项工作，我们需要使用D3 JavaScript图形库。不过这个库只能接受JSON格式的数据。

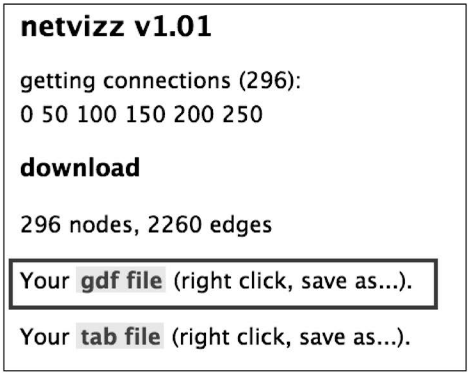
(3) 计算社交网络的度量标准，比如网络大小（也成为度）、网络中两个人之间的最短路径等。在这项工作中我们需要使用Python的`networkx`包。这个包能够接受的数据格式是基于文本的Pajek。

项目的主要目标是演示如何根据需求在不同的数据格式（GDF、Pajek和JSON）之间进行调和，完成从一种数据格式到另一种数据格式的转换。次要目标则是为读者提供足够的示例代码和指导，完成小规模的社交网络分析。

4.4.1 第一步：下载 GDF 格式的 Facebook 数据

在这个步骤中，你需要登录Facebook账号，然后使用Facebook的搜索功能查找netvizz应用，或是直接访问netvizz的地址：https://apps.facebook.com/netvizz/。

在netvizz页面上，点击personal network。这个页面上的start按钮会提供两种格式的下载文件：一种是含有朋友信息以及他们之间关系的GDF格式文件，另一个是制表符分隔值（TSV）格式的统计文件。在这个项目里，我们主要还是关注GDF文件。点击start按钮，在新页面中右键点击GDF文件并把它保存到本地磁盘上，如下图所示。



Facebook的netvizz应用可以让我们以GDF文件格式下载自己的社交网络

在下载文件时最好重新为文件起一个稍微短一些的名字。（我把下载文件命名为personal.gdf并保存在项目所在的目录中。）

4.4.2 第二步：在文本编辑器中查看 GDF 文件

用你的文本编辑器打开下载文件（我使用的是Text Wrangler），然后从以下几个方面观察文件中的内容。

(1) 文件内容分为两个部分：节点和边。

(2) 第一部分的节点信息出现在单词`nodedef`之后。从节点列表中我们可以找出好友清单以及他们之间的基本信息（如性别和Facebook中的ID号）。节点的排列顺序是按照每个人加入Facebook的时间组织起来的。

(3) 第二部分的节点信息表现了好友之间的边或联系。有时候，这些东西也被称为链接。这部分信息都是伴随着单词edgedef出现的。边描述了我的好友中哪些人互为好友。

下面是节点数据的样本：

```
nodedef>name VARCHAR,label VARCHAR,sex VARCHAR,locale
  VARCHAR,agerank INT
  1234,Bugs Bunny,male,en_US,296
  2345,Daffy Duck,male,en_US,295
  3456,Minnie Mouse,female,en_US,294
```

下面是边数据的样本。其中显示了Bugs（1234）和Daffy（2345）互为好友，同时Bugs也是Minnie（3456）的好友：

```
edgedef>node1 VARCHAR,node2 VARCHAR
1234,2345
1234,3456
3456,9876
```

4.4.3　第三步：从 GDF 格式到 JSON 格式的转换

在接下来的任务中我们要利用D3构建一张社交网络的数据展现图。首先，我们需要多看看一些用D3创建社交网络的示例，比如D3的示例集，https://github.com/mbostock/d3/wiki/Gallery和http://christopheviau.com/d3list/。

这些社交网络图的例子全都依赖于JSON数据。每个JSON格式的数据文件都包含了节点和边的信息。下面是其中一份JSON文件的示例：

```
{"nodes": [
  {"name":"Bugs Bunny"},
  {"name":"Daffy Duck"},
  {"name":"Minnie Mouse"}],
  "edges": [
  {"source": 0,"target": 2},
  {"source": 1,"target": 3},
  {"source": 2,"target": 3}]}
```

JSON代码中最为重要的内容就是与GDF格式类似的两大部分：节点和边。节点就是每个人的名字，边则是多组朋友之间的关系。虽然这些信息没有使用Facebook的ID编码，但每组数据都使用了它们在节点列表中的索引信息，其中的索引值始于0。

此时此刻我们是没有JSON文件的，只有GDF文件。那怎么才能得到JSON格式的文件呢？仔细看看GDF文件，会发现里面的内容就好像是两个被黏贴在一起的CSV文件，彼此首尾相连。这一章前面我们就知道，从CSV格式到JSON格式的转换方法不止一种。

因此，我们可以自己动手把GDF转换成CSV文件，然后再把CSV转换成JSON文件。

> 等等，如果D3示例中的JSON数据，与我们下载下来准备用于构建社交网络展现图的JSON文件不一样怎么办？在有的D3社交网络可视化的示例中，你会发现数据中的节点和链接有许多额外的值，比如它们会使用一些额外的属性来表现不同的大小、悬浮效果或颜色的变化，具体示例可以参考http://bl.ocks.org/christophermanning/1625629。这个可视化实现表现的是芝加哥付费政治说客之间的关系。在该示例中，程序代码会根据JSON文件的内容决定代表节点的圆形图案大小，以及鼠标停留在节点上时显示的文字内容。图形效果真的很不错，但实现起来也确实复杂。由于我们的主要目标是数据清洗，所以这一切都可以从简，这些额外的设置不会出现在演示示例中。不过你也不用太担心，我们还是会把D3图表创建出来的！

要把GDF文件转换成我们需要的JSON格式，需要完成以下步骤。

(1) 使用文本编辑器把personal.gdf文件拆分成两个文件，nodes.gdf和links.gdf。

(2) 根据最终要生成的JSON文件格式修改每个文件的标题行：

```
id,name,gender,lang,num
1234,Bugs Bunny,male,en_US,296
2345,Daffy Duck,male,en_US,295
9876,Minnie Mouse,female,en_US,294

source,target
1234,2345
1234,9876
2345,9876
```

(3) 使用工具`csvcut`（之前提到的csvkit中的一部分）从nodes.gdf文件提取第一个和第二个字段，并把结果重定向到一个名为nodesCut.gdf的新文件中：

```
csvcut -c 1,2 nodes.gdf > nodesCut.gdf
```

(4) 现在我们需要为每组边信息指派一个索引值，用来替代原来的Facebook ID值。索引的作用就是通过节点在列表中的位置来定位节点。在尽可能减少重构的基础上，我们需要对数据做一些转换，让它们可以比较容易地满足D3示例中的要求。因此，我们需要把数据从下面的格式：

```
source,target
1234,2345
1234,9876
2345,9876
```

转换成：

```
source,target
0,1
```

```
0,2
1,2
```

下面的Python脚本可以用来自动创建这些索引信息：

```
import csv

# 读取节点信息
with open('nodesCut.gdf', 'r') as nodefile:
    nodereader = csv.reader(nodefile)
    nodeid, name = zip(*nodereader)

# 读取边数据中的原始节点与目标节点
with open('edges.gdf', 'r') as edgefile:
    edgereader = csv.reader(edgefile)
    sourcearray, targetarray = zip(*edgereader)
slist = list(sourcearray)
tlist = list(targetarray)

# 找出原始节点和目标节点所对应的节点索引值
for n,i in enumerate(nodeid):
    for j,s in enumerate(slist):
        if s == i:
            slist[j]=n-1
    for k,t in enumerate(tlist):
        if t == i:
            tlist[k]=n-1
# 以索引值的形式重新编写边信息
with open('edgelistIndex.csv', 'wb') as indexfile:
    iwriter = csv.writer(indexfile)
    for c in range(len(slist)):
        iwriter.writerow([ slist[c], tlist[c]])
```

(5) 现在再回到nodesCut.csv文件中，删除`id`字段：

```
csvcut -c 2 nodesCut.gdf > nodesCutName.gdf
```

(6) 写一个Python脚本把每个文件转换成适用于D3处理的完整JSON文件：

```
import csv
import json

# 读取节点信息
with open('nodesCutName.gdf') as nodefile:
    nodefile_csv = csv.DictReader(nodefile)
    noutput = '['
    ncounter = 0;

    # 处理字典行
    for nrow in nodefile_csv:
        # 查找节点名字中的单引号字符，比如O'Connor
        nrow["name"] = \
        str(nrow["name"]).replace("'","")
        # 在实体数据中间放置一个逗号
```

```
        if ncounter > 0:
            noutput += ','
        noutput += json.dumps(nrow)
        ncounter += 1
    noutput += ']'
    # 将新的数据文件写入磁盘
    f = open('complete.json','w')
    f.write('{')
    f.write('\"nodes\":' )
    f.write(noutput)

# 读取边信息
with open('edgelistIndex.csv') as edgefile:
    edgefile_csv = csv.DictReader(edgefile)
    eoutput = '['
    ecounter = 0;
    # 处理字典行
    for erow in edgefile_csv:
        # 确保数字类型的数据以数字格式保存，而不是字符串格式
        for ekey in erow:
            try:
               erow[ekey] = int(erow[ekey])
            except ValueError:
                # 非int的时候
                pass
        # 在实体数据中间放置一个逗号
        if ecounter > 0:
            eoutput += ','
        eoutput += json.dumps(erow)
        ecounter += 1
    eoutput += ']'

    # 将新文件写入到磁盘中
    f.write(',')
    f.write('\"links\":')
    f.write(eoutput)
    f.write('}')
    f.close()
```

4.4.4　第四步：构建 D3 图

这一节将演示如何使用JSON文件来构建一张D3力导向图。示例代码来自于D3网站。每个节点都用一个圆圈表示，当我们把鼠标放到节点上方的时候，对应的人物名字就会以提示形式显示出来：

```
<!DOCTYPE html>
<!-- 此代码基于https://gist.github.com/mbostock/4062045上的D3力导向图示例 -->

<meta charset="utf-8">
<style>
```

```
.node {
  stroke: #fff;
  stroke-width: 1.5px;
}

.link {
  stroke: #999;
  stroke-opacity: .6;
}

</style>
<body>
<!-- 确保你下载了D3库，并已保存在本地 -->
<script src="d3.min.js"></script>
<script>

var width = 960, height = 500;
var color = d3.scale.category20();
var force = d3.layout.force()
    .charge(-25)
    .linkDistance(30)
    .size([width, height]);

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

d3.json("complete.json", function(error, graph) {
  force
      .nodes(graph.nodes)
      .links(graph.links)
      .start();

  var link = svg.selectAll(".link")
      .data(graph.links)
      .enter().append("line")
      .attr("class", "link")
      .style("stroke-width", function(d) { return Math.sqrt(d.value); });

  var node = svg.selectAll(".node")
      .data(graph.nodes)
      .enter().append("circle")
      .attr("class", "node")
      .attr("r", 5)
      .style("fill", function(d) { return color(d.group); })
      .call(force.drag);

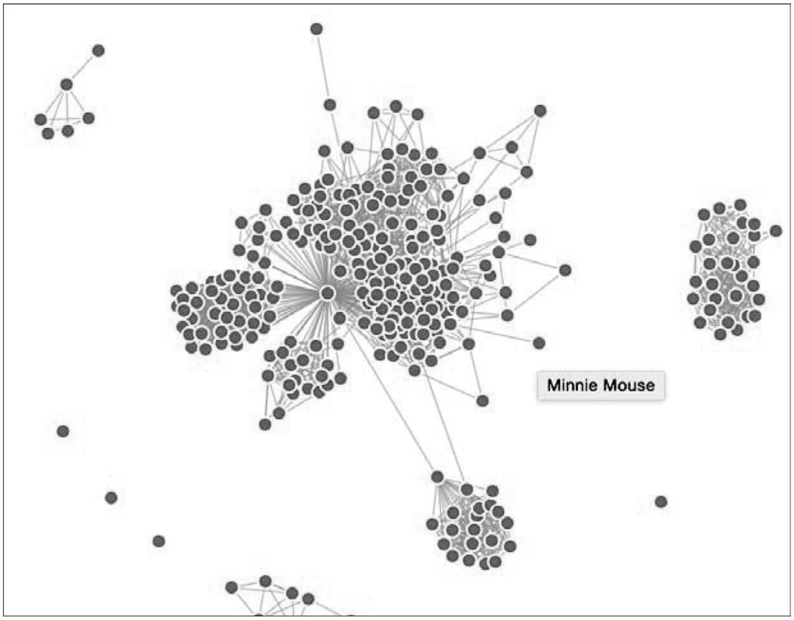
  node.append("title")
      .text(function(d) { return d.name; });

  force.on("tick", function() {
    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
```

```
        .attr("y2", function(d) { return d.target.y; });

    node.attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });
  });
});
</script>
```

下面是社交网络的截图。鼠标正处于其中一个节点的上方，所以从图中我们可以看到这个节点的提示信息（人物名字）。



由D3构建出来的社交网络

4.4.5 第五步：把数据转换成 Pajek 格式

到目前为止，我们已经成功地把GDF文件转换成CSV文件、JSON文件，并用它创建了一张D3图。在接下来的两个步骤里，我们将继续实现下一个目标，把现有的数据转换成可用于社交网络计算的格式。

在这一步，我们将把原始的GDF文件调整成有效的Pajek文件，因为这种格式是社交网络工具networkx所需要的。

> 单词pajek的意思是斯洛维尼亚蜘蛛。社交网络就可以被想象成一个由节点和节点间的连接线组成的大网。

从Facebook的GDF格式到Pajek格式的转换如下：

```
*vertices 296
1234 Bugs_Bunny male en_US 296
2456 Daffy_Duck male en_US 295
9876 Minnie_Mouse female en_US 294
*edges
1234 2456
2456 9876
2456 3456
```

下面是一些关于Pajek文件格式的转换要点。

- 这种文件是通过空格而非逗号分隔的。
- 与GDF文件一样，数据主要分为两大部分，而且这两部分都有特定的标签标注，并以星号（*）开始。这两段数据分别被称为顶点（也称为节点）和边。
- 文件中有一个关于顶点（节点）个数的统计，统计数字位于第一行单词vertices的后面。
- 出现在每个人物名字中的空格字符都由下划线取代。
- 节点数据段中的其余字段都是可选的。

要把GDF文件转换成Pajek格式，我们使用文本编辑器就够了，因为这些修改相对来说还是比较简单的，而且数据量也不是特别大。我们可以像下面那样完成数据清洗工作。

(1) 把原来的GDF文件另保存一份，并重新命名为fbPajek.net（扩展名.net常用来表示Pajek网络文件）。

(2) 把第一行中的内容替换掉。现在文件的内容如下：

```
nodedef>name VARCHAR,label VARCHAR,sex VARCHAR,locale VARCHAR,agerank INT
```

你需要把它修改为：

```
*vertices 296
```

请确保顶点的数目与实际文件中的数目一致。这实际上就是节点的数量，与GDF文件中的每一行信息对应。

(3) 替换文件中的边。现在文件的内容如下：

```
edgedef>node1 VARCHAR,node2 VARCHAR
```

你需要把它修改成：

```
*edges
```

(4) 从第二行开始，我们需要把空格字符全部都替换成下划线。由于文件中只有名字部分包含空格字符，所以我们可以放心地这样做。下面是替换前的内容：

```
1234,Bugs Bunny,male,en_US,296
2456,Daffy Duck,male,en_US,295
3456,Minnie Mouse,female,en_US,294
```

以下是替换后的结果：

```
1234,Bugs_Bunny,male,en_US,296
2456,Daffy_Duck,male,en_US,295
3456,Minnie_Mouse,female,en_US,294
```

(5) 现在可以使用查找与替换功能把所有逗号都替换成空格了。操作完成后节点部分的数据如下：

```
*vertices 296
1234 Bugs_Bunny male en_US 296
2456 Daffy_Duck male en_US 295
3456 Minnie_Mouse female en_US 294
```


边部分的数据结果如下：

```
*edges
1234 2456
2456 9876
2456 3456
```

(6) 最后，使用文本编辑器的查找功能定位名字中含有单引号的Facebook好友。然后通过替换功能将所有的单引号全部替换为空。因此，数据`Cap'n_Crunch`会变成：

```
1998988 Capn_Crunch male en_US 137
```

现在得到的就是清洗后的Pajek格式的文件。

4.4.6 第六步：简单的社交网络分析

讲到这里，我们马上就要使用Python包networks来计算一些简单的社交网络数据。虽然对于社交网络分析（SNA）的讨论已经超出了本书的讨论范围，但我们还是会适当地做一些与比较容易实现的SNA计算。

要完成这个目标，首先要保证安装了networkx包。由于我使用Canopy作为Python编辑器，所以可以使用Package Manager来检查是否安装了networkx。

确认networkx安装之后，我们可以快速地写出一段Python代码来读取Pajek文件，并从Facebook网络结构中输出一些有趣的内容：

```
import networkx as net

# 读取文件
g = net.read_pajek('fb_pajek.net')
```

```
# 图中有多少个节点?
# print len(g)

# 创建一个度值图: 一组名值对, 将节点与网络中边的数量相连结
deg = net.degree(g)
# 对度值图进行排序, 并输出度值最高（社交网络中边数最多）的10个节点
print sorted(deg.iteritems(), key=lambda(k,v): (-v,k))[0:9]
```

我的社交关系输出展现如下。其中列出了前十个节点，每个节点中标注了还有多少其他节点与这个节点相连：

```
[(u'Bambi', 134), (u'Cinderella', 56), (u'Capn_Crunch', 50),
  (u'Bugs_Bunny', 47), (u'Minnie_Mouse', 47), (u'Cruella_Deville',
  46), (u'Alice_Wonderland', 44), (u'Prince_Charming', 42),
  (u'Daffy_Duck', 42)]
```

这里显示`Bambi`与我的`134`个好友有联系，但`Prince_Charming`只与我的`42`个好友有联系。

> 如果你在运行Python时遇到缺少引号的错误，请多检查几次Pajek文件，确保所有的节点标签都不包含空格和其他特殊字符。在前面的清洗中我们已经去除了空格和引号字符,但即便如此,你的好友名字中还是有可能包含别的奇特字符!

当然，你还可以用networkx与D3的可视化功能做些别的有趣的事情，这个示例项目只是让我们了解一下数据清洗过程对于任何成功的大型项目的重要意义。

4.5　小结

在这一章中我们学会了多种把数据从一种格式转换成另一种格式的方法。其中一些技术比较简单，比如直接把文件保存为我们想要的格式，或是通过菜单选项直接输出正确的格式。但有些时候，我们则需要亲自动手编写解决方案。

在许多项目中，包括本章的示例项目，需要不同的清洗步骤，所以我们必须仔细认真地做好清洗计划，并对做过的清洗工作做好记录。networkx和D3都是非常好用的工具，但它们对能够处理的数据格式也是有要求的。同样，虽然我们可以通过netvizz轻松地拿到Facebook数据，但是这些数据的格式比较特殊。因此，能够轻松地从一种文件格式转换成另外一种文件格式，已成为数据科学工作中不可或缺的技能。

在这一章里，我们已经在结构化和非结构化数据之间做了大量的转换练习。但是要怎么样才能清洗诸如无结构化文本这样的杂乱数据呢？

在接下来的第5章，我们会继续丰富我们的数据科学清洗工具箱，学习更多的方法来清洗Web页面。

第 5 章

收集并清洗来自网络的数据

在所有的厨具当中，滤器是最常用和有用的工具之一，它的作用就是在烹饪的过程中将固体从液体中分离出来。在这一章中，我们将为来自网络的数据构建滤器，学习如何利用不同类型的程序找到并保存我们想要的数据，同时去除不需要的内容。

在这一章中，我们将要完成以下几项内容。

- 理解两种不同的HTML页面结构，第一种是行集合，我们可以根据样式匹配其中的内容；第二种是带有节点的树形结构，我们可以利用这些节点来识别并找出想要的数据。
- 尝试使用三种方法来解析页面，一种是使用逐行方法（基于正则表达式的HTML分析技术），另外两种则是使用树形结构方法（Python的BeautifulSoup库和一个叫作Scraper的基于Chrome浏览器的工具）。
- 根据实际的数据情况来实现这三种技术方案；我们将从已经发布的网络论坛消息中去除日期和时间。

5.1 理解 HTML 页面结构

网页就是一个纯文本文件，其中包含了一些特殊的标记元素（有时候称为HTML标签），用来告知浏览器应该如何向用户呈现要显示的内容。例如，如果我们想强调某个单词，那么只需像下面的代码那样把单词用`<em>`标签围住即可：

```
It is <em>very important</em> that you follow these instructions.
```

所有的网页都具备这样的特性；它们本身都是由文本组成的，并且这些文本中会包含许多不同的标签。如果想从网页中提取数据，大抵可以采用两类心智模型。这两类模型分别有着各自的侧重点。在这一节中，我们先对这两类结构化模型做一番介绍，然后在下一节中使用三种不同的工具来进行数据提取演示。

5.1.1　行分隔模型

理解网页的最简单方式就是专注于这一事实：网页上有许多不同的HTML元素/标签，用来在网络上显示页面中的内容。如果我们想从这个简单的模型中抽取自己比较感兴趣的数据，就必须把网页中的文字和嵌套在其中的HTML元素本身当作分隔符。就拿前面的示例来说，我们可以提取标签里面的数据，也可以是标签之前或是标签之后的数据。

在这个模型中，我们需要发挥一下想象力，把网页想象成由超大的无结构文本和一些根据需求组织起来的HTML标签（或别的文本特性，如反复出现的词）共同组成的一个集合，其中的结构化标签可以帮助我们把内容分隔成不同的部分。有了这些分隔标记，从页面提取数据就不会成为难事。

请看下面的HTML页面代码片段，它们取自Django IRC频道的聊天日志。现在让我们想想怎么把HTML元素当成分隔符，提取我们需要的数据：

```
<div id="content">
<h2>Sep 13, 2014</h2>

<a href="/2014/sep/14/">← next day</a> Sep 13, 2014  <a
  href="/2014/sep/12/">previous day →</a>

<ul id="ll">
<li class="le" rel="petisnnake"><a href="#1574618"
  name="1574618">#</a> <span style="color:#b78a0f;8"
  class="username" rel="petisnnake">&lt;petisnnake&gt;</span> i
  didnt know that </li>
  ...
  </ul>
  ...
  </div>
```

在示例文本中，我们可以使用<h2></h2>标签作为分隔符来提取聊天日志中的日期。还可以使用
作为分隔符并找到其中的rel=""来提取聊天人的名字。最后可以提取到
之间由用户向聊天频道发出的消息。

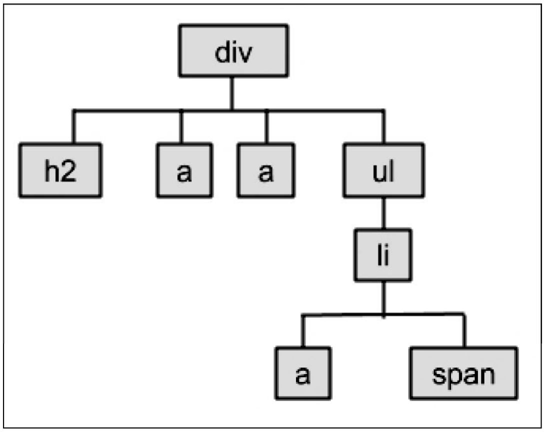
> 这些聊天日志可以从Django IRC日志网站获取，地址为http://django-irc-logs.com。这个网站同时还支持使用关键字搜索日志内容的接口。为了保持代码简洁，部分内容采用省略号（…）代替。

从上面杂乱无章的文本中，我们可以使用分隔符的概念来提取三段数据（日志时间、用户和消息）。

5.1.2　树形结构模型

另一种理解方式是把文本页面想象成一个由HTML元素/标签组成的树形结构，其中每个元素/

标签都与页面上的其他一些标签相关。每个标签都是一个节点，整棵树是由页面中所有节点组成的。在HTML中，如果一个标签出现在另一个标签的内容中，那么内部标签就被称为子标签，外部标签被称为父标签。在之前的IRC聊天日志的例子中，HTML片段展现的树形图如下：



如果把HTML文本想象成树形结构，那么我们就可以使用编程语言来构建这棵树。这样，我们就可以根据元素的名字或是在元素列表中的位置，提取需要的文本数据。例如：

- 我们想要根据标签的名字提取数据（节点`<h2>`中的文本）；
- 我们想要抓取某种标签类型下的所有节点（`<div>`内部的`<ul>`页签下的所有`<li>`节点）；
- 我们想要一个指定元素中的所有属性（`<li>`元素中的`rel`属性）。

在这一章的剩余部分，我们将把心智模型——行分隔模型和树形结构——应用到实际的案例当中。我们将采用三种不同的方法来提取和清洗HTML页面中的数据。

5.2 方法一：Python 和正则表达式

在这一节中，我们使用简单的方法提取HTML页面中的数据。这个方法是基于识别文件中的分隔符的概念，并同时配合正则表达式的使用，最终提取我们需要的数据。

你也许还记得我们在第3章中演练过一些正则表达式吧，那时候我们用的是文本编辑器。其中的一些概念与本章即将描述的都是相通的，只是这次要使用Python来替代我们之前使用的文本编辑器，采用编程的形式来查找匹配的文本并完成数据的提取工作。

在开始实践之前还有一件事我们得铭记于心，即正则表达式这种实现方案虽然非常容易理解，但也有很大的局限性，具体取决于项目的的细节。在这一节的最后我们会对相关的局限性做出详细的说明。

5.2.1　第一步：查找并保存实验用的 Web 文件

在这个例子中，我们将要抓取前面Django项目中提到的IRC聊天记录。这些文件都是公开的，并且结构也比较常规，所以比较适合我们这个项目使用。进入Django IRC日志归档网站http://django-irc-logs.com/，选择一个你认为对你比较有意义的日期。进入所选日期对应的目标页面后，把文件保存到你的本地工作目录。下载完成后你会得到一个以.html结尾的文件。

5.2.2　第二步：观察文件内容并判定有价值的数据

我们在第2章中已经学过，.html文件是纯文本文件，而通过第3章的学习我们已经能够用文本编辑器查看文本文件，所以这一步的目标很容易实现。只需要用文本编辑器打开HTML文件，然后仔细查看其中的内容即可。但是有哪些内容可以提取呢？

在观察过程中，我确实发现了几项可以提取的内容。从每一条聊天记录的备注内容中可以看出，里面有行号、用户名、备注这些信息。接下来我们得为抽取这三项内容做一些准备了。

下面截图显示的是在文本编辑器中打开的HTML文件。当时我打开了折行功能，因为有些行的内容实在是太长了（在Text Wrangler中可以通过View | Text Display | Soft Wrap Text打开这个功能）。在29行上下我们可以看到聊天记录的开始部分，每一行都包括我们想要的那三项内容。

```
29  <ul id="ll">
30  <li class="le" rel="petisnnake"><a href="#1574618" name="1574618">#</a> <span
…   style="color:#b78a0f;8" class="username" rel="petisnnake">&lt;petisnnake&gt;</span> i
…   didnt know that </li>
31  <li class="le" rel="dshap"><a href="#1574619" name="1574619">#</a> <span
…   style="color:#b93d98;8" class="username" rel="dshap">&lt;dshap&gt;</span> FunkyBob: ahh,
…   hmm, i wonder if there's a way to do it in my QuerySet subclass so i'm not creating a new
…   manager subclass *only* for get_queryset to do the intiial filtering </li>
```

接下来的工作就是从每一行中找出看起来具有相同特点的内容，这样就可以从聊天记录里抓取我们所需的那三项内容了。从文本内容中我们可以发现，只要遵循下面的规则，就可以在改动最小的情况下准确地抽取每一项数据。

- 我们需要抽取的三项数据都可以在标签`<li>`内找到，并且这个标签全部处于标签`<ul id="ll">`内部。每一个`<li>`代表一条聊天消息记录。
- 在消息中，行号信息可以从两个位置找出：一处是字符串`<a href="#`之后，另一处是`name`属性后面的双引号之间。在示例中，第一个出现的行号是`1574618`。
- 属性`username`可以在三个地方找到，第一处是标签`li class="le"`的`rel`属性值，第二处是标签`span`中的`rel`属性，最后一处则是处于`<`和`>`之间。在示例中，第一个`username`的值是`petisnnake`。
- 消息信息处于标签`</span>`和`</li>`之间。在例子中，第一条消息是`i didnt know that`。

数据的查找规则已经有了，那就开始写程序吧。

5.2.3 第三步：编写 Python 程序把数据保存到 CSV 文件中

在这里我们编写了下面的代码，用它来打开一个指定的IRC日志文件，文件的格式与我们之前描述的一样。利用这份代码我们可以分别解析出所需的三项数据并把它们输出到CSV文件：

```
import re
import io

row = []

infile  = io.open('django13-sept-2014.html', 'r', encoding='utf8')
outfile = io.open('django13-sept-2014.csv', 'a+', encoding='utf8')
for line in infile:
    pattern = re.compile(ur'<li class=\"le\" rel=\"(.+?)\"><a
        href=\"#(.+?)\" name=\"(.+?)<\/span> (.+?)</li>', re.UNICODE)
    if pattern.search(line):
        username = pattern.search(line).group(1)
        linenum = pattern.search(line).group(2)
        message = pattern.search(line).group(4)
        row.append(linenum)
        row.append(username)
        row.append(message)
        outfile.write(', '.join(row))
        outfile.write(u'\n')
        row = []
infile.close()
```


这段代码最复杂的地方就是`pattern`那一行。文件中的每一行文本都会与该行代码构建出来的样式匹配进行比较。

> 注意，网站的开发者可以随时改变HTML结构，所以我们是冒着风险干这件事的，因为说不定哪天我们精心编写的样式就无法工作了。实际上，在编写本书的几个月时间里，这个页面的HTML至少发生过一次变化！

每组匹配的数据样式都是`.+?`。这样的数据一共有五组，其中三组是我们有兴趣的数据（用户名、行号和消息），另外两组对我们没有什么用处，可以直接丢弃。页面中的其余内容也都可以丢弃，因为它们与我们设计的样式无法匹配。我们的程序就像一个筛子一样，当中恰好留了三个功能漏点。有用的数据都会从这些漏点中过滤下来，留下的都是没有用的数据。

5.2.4 第四步：查看文件并确认清洗结果

用文本编辑器打开新创建的CSV文件，这时我们看到的前几行内容应该与下面一样：

```
1574618, petisnnake, i didnt know that
1574619, dshap, FunkyBob: ahh, hmm, i wonder if there's a way to do
it in my QuerySet subclass so i'm not creating a new manager subclass
```

```
*only* for get_queryset to do the intiial filtering
1574620, petisnnake, haven used Django since 1.5
```

结果看起来还不赖。不过有件事你可能已经注意到了，第三列文本没有使用封闭字符。由于我们把逗号作为分隔符号使用，这下我们得面对新的问题了。第三列文本中的逗号该怎么处理呢？如果这个问题让你苦恼的话，你可以选择用引号把第三列内容封闭起来，或者是改用制表符来作为分隔符号。要想改变分隔符号，我们需要对第一个`outfile.write()`语句做一些修改，把当中连接用的逗号全部替换成`\t`（制表符）。在处理中，你还可以使用函数`ltrim()`来清洗消息中包含的多余空白字符。

5.2.5　使用正则表达式解析 HTML 的局限性

正则表达式这个方案乍看起来挺简单的，但它是有局限性的。首先，对于数据清洗工作者来说，设计正则表达式样式可能就是一件苦差事。你得花费大量的时间和精力来调试样式、编写冗长的说明文档。所以，为了辅助正则表达式的开发，我极力推荐使用像http://Pythex.org这样的正则表达式测试工具，当然，你也可以用搜索引擎找一款更适合你自己使用的测试工具。如果你使用的语言是Python的话，一定要指明你想要的是Python正则表达式测试工具。

接下来，你应该事先搞清楚一件事，正则表达式是完全依赖网页结构的。因此，如果你打算从网站上收集数据，那么你今天编写的正则表达式很有可能明天就无法工作了。表达式样式只有在页面布局没有发生变化的时候才能正常工作。或许刚刚在两个标签之间加入的一个空格字符就能让整个正则表达式失效，但分析起来却是困难重重。还有一点需要谨记的是，你无法干预网站的变化，因为要收集的大部分数据不是源于你自己的网站！

最后要讲的是，很多情况下无法使用正则表达式来精确地匹配HTML结构的。正则表达式虽然强大，但它并非完美得无懈可击。对于这个问题，我推荐你参考著名的Stack Overflow上一个超过4000次点赞的答案：http://stackoverflow.com/questions/1732348/。在这个答案中，作者幽默地表达了许多程序员的失望之情，他们曾经一次次尝试解释为什么正则表达式不是解析无规则的频繁变化的HTML页面的最佳方案。

5.3　方法二：Python 和 BeautifulSoup

因为正则表达式有诸多局限性，所以我们必须得用更多的工具来进行补充。在这里，我们将使用Python的BeautifulSoup库来解析树形结构的HTML页面。

5.3.1　第一步：找到并保存实验用的文件

在这一步中，我们还使用方法一中的那个文件：来自Django IRC频道的文件。我们还是提取相同的三个内容。这样做很容易比较出这两种方法的差异。

5.3.2 第二步：安装 BeautifulSoup

BeautifulSoup目前已经发展到第四版。这个版本同时兼容Python 2.7和Python 3。

> 如果你使用的是Enthought Canopy Python环境，在Canopy Terminal里运行`pip install beautifulsoup4`就可以安装BeautifulSoup。

5.3.3 第三步：编写抽取数据用的 Python 程序

我们需要的三项内容可以从`li`标签集合中找到，更准确地说是那些带有`class="le"`样式的标签。在这个文件中虽然再没有别的`li`标签了，但为了谨慎起见我们在操作时还是应该尽可能地指明具体信息。下面的列表描述了在解析树形结构时我们需要获取的内容以及它们的所在位置。

- 我们可以从`li`标签中的`rel`属性提取`username`值。
- 我们可以从标签中的`name`属性提取`linenum`值。这个标签指的是`li`内部的第一个`a`标签。

> 请记住，数组索引是从零开始的，所以我们需要使用0获取第一个项目的内容。

  在BeautifulSoup中，标签的内容就是被解析的树形结构的标签的内部项目，在有些包中被称为子项。

- 我们可以把`message`当作`li`标签的第四项内容（也就是数组中的内容[3]）来处理。还有一点需要注意的是，每条消息的前面都有一个打头的空白字符，我们需要在保存数据之前把它处理掉。

下面是解析树形结构的Python代码：

```
from bs4 import BeautifulSoup
import io

infile  = io.open('django13-sept-2014.html', 'r', encoding='utf8')
outfile = io.open('django13-sept-2014.csv', 'a+', encoding='utf8')
soup = BeautifulSoup(infile)

row = []
allLines = soup.findAll("li","le")
for line in allLines:
    username = line['rel']
    linenum = line.contents[0]['name']
    message = line.contents[3].lstrip()
    row.append(linenum)
    row.append(username)
    row.append(message)
```

```
        outfile.write(', '.join(row))
        outfile.write(u'\n')
        row = []
    infile.close()
```

5.3.4　第四步：查看文件并确认清洗结果

用文本编辑器打开新创建的CSV文件，从前几行数据中我们可以发现，这基本上与第一种方法产生的结果一样：

```
1574618, petisnnake, i didnt know that
1574619, dshap, FunkyBob: ahh, hmm, i wonder if there's a way to do
it in my QuerySet subclass so i'm not creating a new manager subclass
*only* for get_queryset to do the intiial filtering
1574620, petisnnake, haven used Django since 1.5
```

与正则表达式方案类似，如果你对最后一列中的逗号分隔符不放心的话，可以用双引号把其中的文本封闭起来，或者是改用制表符作为字段分隔符。

5.4　方法三：Chrome Scraper

如果你不想用程序解析数据的话，还有一些基于浏览器的工具可以让你从树形结构中抽取数据。我认为最简单省事的方案就是使用Chrome应用扩展中一个叫作Scraper的工具，它的开发者名为mnmldave（真名：Dave Heaton）。

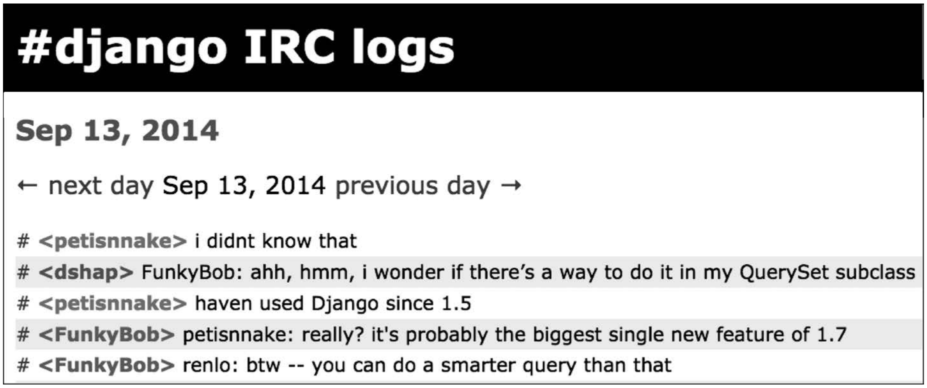
5.4.1　第一步：安装 Chrome 扩展 Scraper

如果你还没有安装Chrome浏览器的话，那就先下载一个。然后再确认一下你是不是安装了正确版本的Scraper，因为名字类似的扩展插件不止一个。我推荐直接从开发者本人的GitHub站点获取这个扩展插件，地址为http://mnmldave.github.io/scraper/。通过这种途径获取的程序肯定没有问题，这样你就不用在Chrome商店中到处找了。从http://mmldave.github.io/scraper点击链接安装完扩展插件后，重新启动浏览器。

5.4.2　第二步：从网站上收集数据

现在可以用你的浏览器访问我们之前两个数据提取实验所使用的URL，随便访问一个Django IRC日志。我使用的是2014年9月13日的数据，所以我的访问地址是http://django-irc-logs.com/2014/sep/13/。

在本书编写之时，页面在我的浏览器中看起来是下面这个样子：



IRC日志中已经包含了我们需要提取的三项内容：

- 行号信息（从之前的两个实验中我们可以得知，链接内容里符号#之后的文本就是行号）
- 用户名（符号<和>之间的文本）
- 消息信息

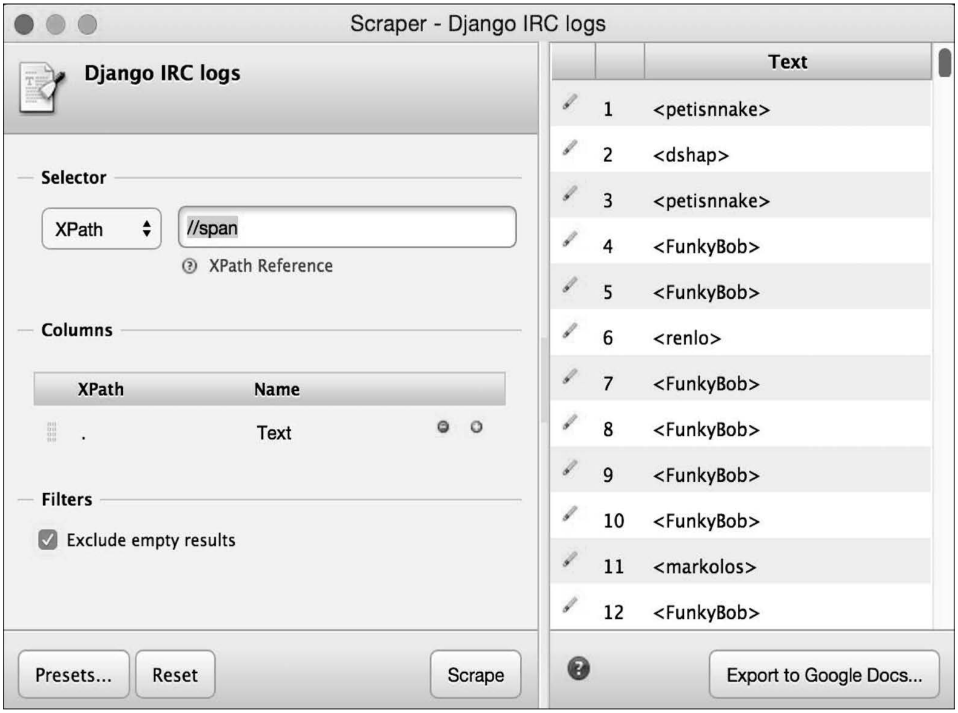
Scraper可以依次把那些要提取的数据进行高亮显示，并把结果输出到Google Spreadsheet中去，之后我们可以把这些输出结果重新整理到一张工作表里，并以CSV格式（或是别的格式）输出。下面是具体的操作步骤。

(1) 使用鼠标高亮选中你想要抓取的数据。

(2) 点击鼠标右键并从菜单中选择Scrape similar...。从下面的截图中可以看出，我选择用户名petisnnake作为抓取对象。



(3) 选择Scrape similar之后，工具显示出一个新的窗口，其中包括了页面中其他相似的项目。下面的截图显示了Scraper找出的完整用户名列表。



Scraper根据示例中的用户名找出了所有类似的项目。

(4) 在窗口的底部有一个名为Export to Google Docs...的按钮。这里需要注意的是，在不同的设置条件下，你可能需要点击同意之后才可以让Scraper访问Google Docs。

5.4.3　第三步：清洗数据

当我们把所有需要的数据元素都从页面中取出来并放到Google Docs后，就可以将它们组织到一个文件里并做最后的清洗工作了。下面是行号信息抽取之后的样子，而此时我们还没有对它们进行清洗。

	A	B
1	Link	URL
2	#	#1574618
3	#	#1574619
4	#	#1574620
5	#	#1574621
6	#	#1574622

对于A列中的内容我们并不关心，而且打头的符号#也不是我们想要的。用户名和每行的消

息数据看起来也差不多——其中大部分数据都是有效的，但我们还是得先移除不需要的符号，然后把所有的内容都集中到一个独立的Google Spreadsheet中去。

使用我们在第3章中学到的查找与替换技术（即删除符号#、<和>之后，把所有的行粘贴到一个新的工作表中），可以得到像下面一样干净的数据集：

	A	B	C
1	1574618	petisnnake	i didnt know that
2	1574619	dshap	FunkyBob: ahh, hmm, i wonder if there's a way to do it in my QuerySet
3	1574620	petisnnake	haven used Django since 1.5
4	1574621	FunkyBob	petisnnake: really? it's probably the biggest single new feature of 1.7
5	1574622	FunkyBob	renlo: btw -- you can do a smarter query than that

Scraper非常适合从网页中抓取少量数据。它有一个便捷的Google Spreadsheets接口，如果你不想编写程序的话，这不失为一个快捷的解决方案。在接下来的一节中，我们将迎接一个更大的项目。它非常复杂，可能把这一章中的多个概念组合到一起才能形成一个完整的解决方案。

5.5　示例项目：从电子邮件和论坛中抽取数据

Django IRC日志项目非常简单。它的目的就是向你展示三种常见的HTML页面数据抽取技术之间的差异。我们需要抽取的数据包括行号、用户名和IRC聊天消息，找到这些数据并不难，几乎也不需要什么额外的清洗操作。而在即将开始的新项目中，概念上和之前的项目还是有相似之处的，但我们得把数据抽取思路扩展到HTML格式之外的两种Web中常见的半结构文本格式：Web中的电子邮件消息和基于网页的论坛消息。

5.5.1　项目背景

我最近做了一项关于社交媒体如何为软件技术提供支持的研究。具体说来，我想知道，是不是应该让某些API和框架的软件开发组织把他们对开发人员的支持转移到Stack Overflow上去，或者他们是不是应该继续使用传统的媒体模式，比如电子邮件和网络论坛。为了完成这项研究，我特意比较了软件开发人员获取API问题答案的时间，其中有的是通过Stack Overflow，有的是通过传统的社交媒体，比如网络论坛和电子邮件组。

在这个项目里，我们将完成这个问题的一小部分。我们会下载两种不同类型的原始数据，分别代表了不同的传统社交媒体：网络论坛的HTML文件和Google Groups的电子邮件。我们会编写Python代码来提取这两个论坛消息里所包含的日期和时间。之后再整理出哪些消息是用于回复其他消息的，并对回复的间隔时间做一个简单的汇总统计。

> 如果你想知道为什么我们不抽取Stack Overflow上的数据样本来回答刚才提出的问题，等到第9章你就明白了。在那一章我们会专门创建并清洗Stack Overflow数据库。

这个项目被划分为两个部分。在第一个部分中，我们从Google Groups的一个项目的归档邮件中抽取数据，而在第二部分中，我们从另一个项目的HTML文件抽取数据。

5.5.2　第一部分：清洗来自 Google Groups 电子邮件的数据

现在还有许多软件公司在使用传统的电子邮件列表，或是同时使用邮件与网络论坛来为他们的产品提供技术支持。Google Groups就是这种服务的一种流行应用。用户可以向组内发送邮件，或是通过网络浏览器阅读与搜索邮件消息。但是，有的公司已经把技术支持搬离了Google Groups（包括Google本身的一些产品），取而代之的是使用Stack Overflow。数据库产品Google BigQuery就是使用Stack Overflow的组织之一。

1. 第一步：收集Google Groups消息

为了研究BigQuery在Google Group上的问题响应时间，我特意为组内所有的回复创建了一份URL清单列表。你可以从https://github.com/megansquire/stackpaper2015/blob/master/BigQuery-GGurls.txt找到完整的列表内容。

有了目标URL列表之后，我们就可以编写Python程序来下载URL中所涉及的所有电子邮件，并把它们保存到磁盘中。在下面的程序里，我把URL清单列表保存在一个名为GGurls.txt的文件里。程序引用了`time`库，这样就可以在请求Google Groups服务器时使用`sleep()`函数稍作停歇：

```
import urllib2
import time

with open('GGurls.txt', 'r') as f:
    urls = []
    for url in f:
        urls.append(url.strip())

currentFileNum = 1
for url in urls:
    print("Downloading: {0} Number: {1}".format(url, currentFileNum))
    time.sleep(2)
    htmlFile = urllib2.urlopen(url)
    urlFile = open("msg%d.txt" %currentFileNum,'wb')
    urlFile.write(htmlFile.read())
    urlFile.close()
    currentFileNum = currentFileNum +1
```

程序运行之后一共在磁盘上生成了667个文件。

2. 第二步：从Google Groups消息中抽取数据

现在我们已经有了667组电子邮件消息，每组消息都以独立文件的形式存在。接下来的任务就是编写一个程序每次读取一个文件，并用本章中学到的技术从文件中提取我们需要的信息。无论从哪个邮件中我们都可以发现很多头部信息，其中存储了电子邮件信息或是相关的元数据。现在让我们快速地浏览一下三段头部信息，找出我们需要的原数据：

```
In-Reply-To: <ab71b72a-ef9b-4484-b0cc-a72ecb2a3b85@r9g2000yqd.
googlegroups.com>
Date: Mon, 30 Apr 2012 10:33:18 -0700
Message-ID: <CA+qSDkQ4JB+Cn7HNjmtLOqqkbJnyBu=Z1Ocs5-dTe5cN9UEPyA@mail.
gmail.com>
```

所有的消息都有`Message-ID`和`Date`，但`In-Reply-To`只会在回复消息中出现。`In-Reply-To`的值必须是另一条消息的`Message-ID`。

下面的代码演示的是一种基于正则表达式的解决方案，它能够抽取`Date`、`Message-ID`和`In-Reply-To`值，并创建出原始消息与回复消息的列表。然后计算出消息和它们对应的回复消息之间的时间差。

5

```
import os
import re
import email.utils
import time
import datetime
import numpy

originals = {}
replies = {}
timelist = []

for filename in os.listdir(os.getcwd()):
    if filename.endswith(".txt"):
        f=open(filename, 'r')
        i=''
        m=''
        d=''
        for line in f:
            irt = re.search('(In\-Reply\-To: <)(.+?)@', line)
            mid = re.search('(Message\-ID: <)(.+?)@', line)
            dt = re.search('(Date: )(.+?)\r', line)
            if irt:
                i= irt.group(2)
            if mid:
                m= mid.group(2)
            if dt:
                d= dt.group(2)
        f.close()
        if i and d:
            replies[i] = d
```

```
        if m and d:
            originals[m] = d

for (messageid, origdate) in originals.items():
    try:
        if replies[messageid]:
            replydate = replies[messageid]
            try:
                parseddate = email.utils.parsedate(origdate)
                parsedreply = email.utils.parsedate(replydate)
            except:
                pass
            try:
                # 这个地方有时会生成错误的时间值
                timeddate = time.mktime(parseddate)
                timedreply = time.mktime(parsedreply)
            except:
                pass
            try:
                dtdate = datetime.datetime.fromtimestamp(timeddate)
                dtreply = datetime.datetime.fromtimestamp(timedreply)
            except:
                pass
            try:
                difference = dtreply - dtdate
                totalseconds = difference.total_seconds()
                timeinhours =  (difference.days*86400+difference.seconds)/3600
                # 这个地方应该处理负数时间的
                # 或许用时区处理比较合适，暂时还是算了吧
                if timeinhours > 1:
                    # 打印小时数
                    timelist.append(timeinhours)
            except:
                pass
    except:
        pass

print numpy.mean(timelist)
print numpy.std(timelist)
print numpy.median(timelist)
```

从代码中可以看出，最开始的for循环会依次提取我们所需的三种信息。（这段程序不会把提取出来的信息存放到一个独立的文件或是磁盘中，如果有需要的话你可以自己实现这个功能。）它创建了两个重要的列表。

- originals[]是包含原始消息的列表。这些消息都是最初提出的问题。
- replies[]是包含回复消息的列表。这些消息都是针对最初提出的问题的回复。

第二段for循环处理会对原始消息列表中的每条消息做进一步处理，如果原始消息有对应的回复消息，就尝试找出经过多长时间回复消息才被发出。我们会为回复时间做出一个新的记录

列表。

(1) 数据提取代码

在这一章中，我们最为关心的代码是数据清洗和数据抽取部分，所以直接阅读下面的代码就可以了。在这里，程序会逐行处理电子邮件中的内容，目的是找出关键的三个邮件头部信息：`In-Reply-To`、`Message-ID`和`Date`。代码中使用正则表达式来搜索与组织数据，就像我们在前面的方法一中那样，对头部信息进行了拆分并提取出相应的数据：

```
for line in f:
    irt = re.search('(In\-Reply\-To: <)(.+?)@', line)
    mid = re.search('(Message\-ID: <)(.+?)@', line)
    dt = re.search('(Date: )(.+?)\r', line)
    if irt:
        i = irt.group(2)
    if mid:
        m = mid.group(2)
    if dt:
        d = dt.group(2)
```

我们为什么要用正则表达式来代替树形结构的解析方法呢？原因主要有以下两个。

(a) 因为下载下来的电子邮件不是HTML格式，所以我们不能使用父子关系的树形结构来描述文件内容。因此，像BeautifulSoup这样基于树形结构的方案是行不通的。

(b) 因为电子邮件头部信息的结构都是固定的，并且内容也是可以预测的（尤其是我们需要的那三条数据），所以我们可以采用正则表达式来完成这个处理。

(2) 程序输出

程序的输出就是打印出三个数字，表示在该Google Group中以小时为计算单位的情况下，其均方差、标准差和中位差分别是多少。下面是代码在我的环境中运行出来的结果：

```
178.911877395
876.102630872
18.0
```

这意味着在BigQuery Google Group上对一条消息的平均响应时间是18小时。现在让我们再研究一下怎么从另一种数据源——网络论坛——中提取类似的数据。你觉得网络论坛中的消息响应会怎么样呢？更快、更慢，还是跟Google Group上一样？

5.5.3　第二部分：清洗来自网络论坛的数据

在这个项目中我们要研究的网络论坛来自一家名为DocuSign的公司。他们也把对开发者的支持转移到Stack Overflow上了，但同时对之前基于网络的开发者论坛做了归档处理，并且可以在

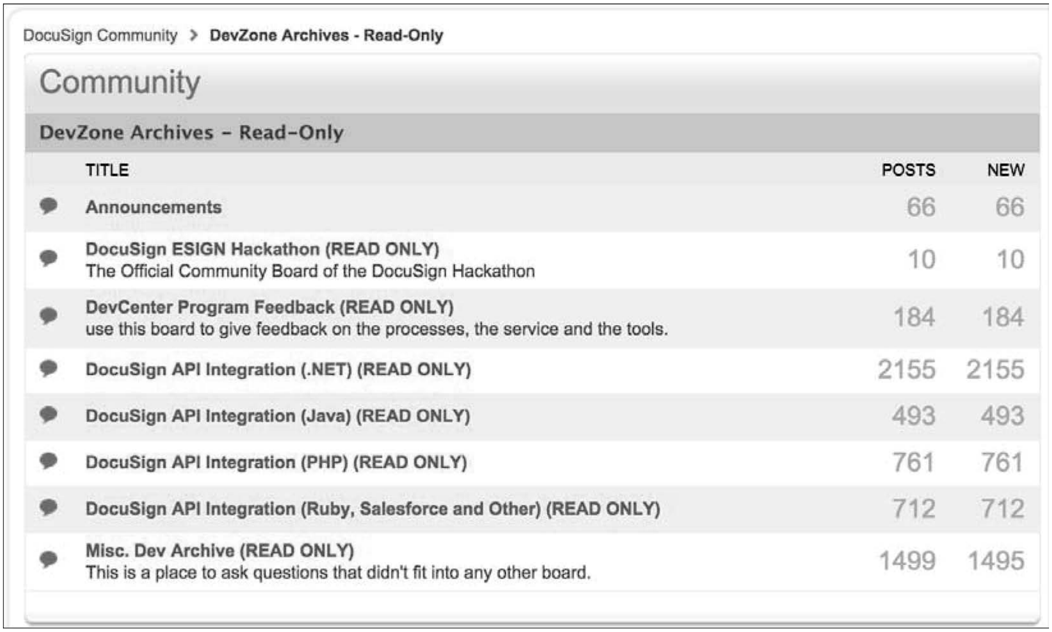
线访问。当时我正在他们的网站上闲游，直到发现了下载旧论坛消息的方法。这里演示的处理方法比Google Groups的例子要多，比如数据的自动化收集。

1. 第一步：收集指向HTML文件的RSS信息

DocuSign开发者论坛中有几千条消息。我们可以组织一份URL列表来指向所有的消息或是讨论主题，这样就可以用代码实现自动下载，更高效地提取回复时间。

要实现这个功能，首先需要的是拿到所有讨论主题的URL。我发现DocuSign在Dev-Zone开发者站点上的归档文件地址为https://community.docusign.com/t5/DevZone-Archives-Read-Only/ct-p/dev_zone。

网站看起来和下面的浏览器截图一样：



我们当然不想挨个点击论坛中的每一个链接，然后手工把每条消息保存下来。要是那样做的话，差不多得花一辈子时间，太烦了。难道就没有别的什么好办法了吗？

DocuSign网站的帮助页面已经表明，通过下载Really Simple Syndication（RSS）文件可以获取每个论坛中最新发布的主题和消息。这样的话就可以利用RSS文件自动收集网站上各种讨论话题的URL信息。而我们最为关注的RSS文件只有开发者支持论坛（并非广告或销售论坛）。这些RSS文件可以从下面的地址获取：

- https://community.docusign.com/docusign/rss/board?board.id=upcoming_releases

- https://community.docusign.com/docusign/rss/board?board.id=DocuSign_Developer_Connection
- https://community.docusign.com/docusign/rss/board?board.id=Electronic_Signature_API
- https://community.docusign.com/docusign/rss/board?board.id=Java
- https://community.docusign.com/docusign/rss/board?board.id=php_api
- https://community.docusign.com/docusign/rss/board?board.id=dev_other
- https://community.docusign.com/docusign/rss/board?board.id=Ask_A_Development_Question_Board

我们可以通过浏览器访问列表中的每一个URL（或者只访问其中的一个）。由于文件是RSS格式的，所以看起来像是带有标签的半结构化文本文件，与HTML比较类似。把访问到的RSS文件保存到本地系统并为每个文件添加一个.rss文件扩展名。在整个处理结束的时候，你应该拿到至少七个RSS文件，每一个文件能找到与上面列表中相对应的URL。

每个RSS文件的内容都是关于论坛中各个讨论话题的元数据，当中包括一些在这个阶段我们迫切需要的数据：每个讨论话题对应的URL地址。用文本编辑器随便打开其中一个RSS文件，你就可以定位到我们需要的URL。具体形式看起来应该跟下面的例子差不多，从文件的内部可以看出，每个讨论话题都含有这样的数据：

```
<guid>http://community.docusign.com/t5/Misc-Dev-Archive-READ-ONLY/
Re-Custom-CheckBox-Tabs-not-marked-when-setting-value-to-quot-X/m-
p/28884#M1674</guid>
```

现在我们可以编写程序来遍历每一个RSS文件，从而找出与之关联的URL数据，接着访问这些地址，提取出我们希望得到的回复时间。在下一节中，我们会把这个过程划分为一系列小步骤，并在后面使用程序来演示如何完成整个工作。

2. 第二步：从RSS中提取URL，收集并解析HTML

在这一步中，我们将编写一个程序来完成以下步骤。

(1) 打开我们在第一步中保存下来的每一个RSS文件。

(2) 每当碰到`<guid>`和`</guid>`的标签组合的时候，就提取其中的URL信息并把它添加到一个列表中。

(3) 根据列表中的每一个URL，下载其对应的HTML文件。

(4) 读取HTML文件内容，并抽取原始消息的发布时间和对应的每一条回复消息的回复时间。

(5) 计算平均差、中位差和标准差，这跟我们当时在第一部分中做的差不多。

下面的Python代码会完成这些步骤。看完代码之后我们将详细说明数据抽取部分的内容：

```
import os
import re
import urllib2
import datetime
import numpy

alllinks = []
timelist = []
for filename in os.listdir(os.getcwd()):
    if filename.endswith('.rss'):
        f = open(filename, 'r')
        linktext = ''
        linkurl = ''
        for line in f:
            # 找出讨论主题的的URL
            linktext = re.search('(<guid>)(.+?)(<\/guid>)', line)

            if linktext:
                linkurl= linktext.group(2)
                alllinks.append(linkurl)
        f.close()

mainmessage = ''
reply = ''
maindateobj = datetime.datetime.today()
replydateobj = datetime.datetime.today()
for item in alllinks:
    print "==="
    print "working on thread\n" + item
    response = urllib2.urlopen(item)
    html = response.read()
    # 定义一个用于匹配时间戳的正则表达式样式
    tuples = re.findall('lia-message-posted-on\">\s+<span
class=\"local-date\">\\xe2\\x80\\x8e(.*?)<\/span>\s+<span
class=\"local-time\">([\w:\sAM|PM]+)<\/span>', html)
    mainmessage = tuples[0]
    if len(tuples) > 1:
        reply = tuples[1]
    if mainmessage:
        print "main: "
        maindateasstr = mainmessage[0] + " " + mainmessage[1]
        print maindateasstr
        maindateobj = datetime.datetime.strptime(maindateasstr,
'%m-%d-%Y %I:%M %p')
    if reply:
        print "reply: "
        replydateasstr = reply[0] + " " + reply[1]
        print replydateasstr
        replydateobj = datetime.datetime.strptime(replydateasstr,
'%m-%d-%Y %I:%M %p')

        # 只针对有回复的数据进行时间差计算
        difference = replydateobj - maindateobj
        totalseconds = difference.total_seconds()
```

```
        timeinhours =  (difference.days*86400+difference.seconds)/3600
        if timeinhours > 1:
            print timeinhours
            timelist.append(timeinhours)

print "when all is said and done, in hours:"
print numpy.mean(timelist)
print numpy.std(timelist)
print numpy.median(timelist)
```

(1) 程序状态

在程序工作的过程中，它会打印出一些状态信息，这样我们就可以知道它是否还处在工作的状态中。状态信息与下面的内容类似，并且每次从RSS文件中找到URL时都会有这样的信息输出：

```
===
working on thread
http://community.docusign.com/t5/Misc-Dev-Archive-READ-ONLY/Can-you-
disable-the-Echosign-notification-in-Adobe-Reader/m-p/21473#M1156
main:
06-21-2013 08:09 AM
reply:
06-24-2013 10:34 AM
74
```


这里的74表示在当前这个话题中，第一条提问消息的发布时间与第一条回复消息的发布时间经过四舍五入之后的差值（约为三天又两个小时）。

(2) 程序输出

在得到结论的时候，程序会分别打印出平均差、标准差和以小时为单位的平均回复时间，这与我们在第一部分的Google Groups程序中做的一样：

```
when all is said and done, in hours:
695.009009009
2506.66701108
20.0
```

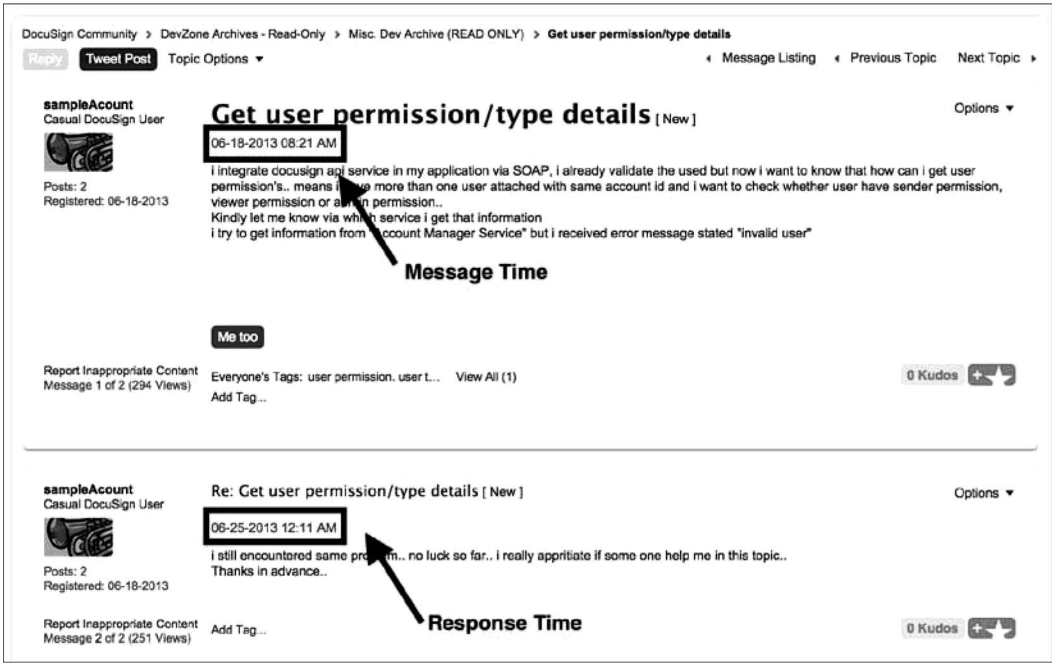
看起来DocuSign论坛上的回复时间比Google Groups还要慢一些。Google Groups需要18小时比，它则需要花费20个小时，不过好在这两个数字还是在比较接近的范围内。你的实验结果可能会有所不同，这是因为总有新的消息不断地加入实验文件。

(3) 数据提取

我们的重点是数据提取。让我们进一步看看程序到底做了些什么。下面是最为重要的代码：

```
tuples = re.findall('lia-message-posted-on\">\s+<span class=\"local-
date\">\\xe2\\x80\\x8e(.*?)<\/span>\s+<span class=\"local-
time\">([\w:\sAM|PM]+)<\/span>', html)
```

就像前面的那些例子一样，这段代码也是依靠正则表达式实现的。但这里的表达式看起来乱得一塌糊涂。也许我们用BeautifulSoup会有所改善吧？那就让我们先看一下原始的HTML内容，这样就可以充分理解这段代码的含义，并能更好地判断是否应该换一种实现方式。下面的截图显示的是页面在浏览器中的样子。我们需要的时间信息在截图中加上了注释：



底层的HTML又是什么样子呢？那正是我们的程序需要解析的。从HTML的内容中我们可以发现，原始消息的日期信息分布在几处不同的地方，但原始消息和回复消息的日期和时间组合只在页面上打印了一次。下面是HTML的内容（为了方便查看，HTML经过简单的压缩处理，并移除了一些多余的空行）：

```
<p class="lia-message-dates lia-message-post-date lia-component-post-
date-last-edited" class="lia-message-dates lia-message-post-date">
<span class="DateTime lia-message-posted-on lia-component-common-
widget-date" class="DateTime lia-message-posted-on">
<span class="local-date">06-18-2013</span>
<span class="local-time">08:21 AM</span>

<p class="lia-message-dates lia-message-post-date lia-component-post-
date-last-edited" class="lia-message-dates lia-message-post-date">
<span class="DateTime lia-message-posted-on lia-component-common-
widget-date" class="DateTime lia-message-posted-on">
<span class="local-date">06-25-2013</span>
<span class="local-time">12:11 AM</span>
```

结果内容明显可以用正则表达式来解决，只要我们编写一个正则表达式样式就可以一次性找到这两种类型的消息。在代码里，我们把第一次找到的内容当作原始消息处理，而接下来的则当作回复消息处理，请看下面的代码：

```
mainmessage = tuples[0]
if len(tuples) > 1:
    reply = tuples[1]
```

我们本来也可以采用BeautifulSoup这样基于树形结构的解决方案，但这就不得不处理两套一模一样的日期组合，因为它们对应的span标签样式是完全相同的，即使我们将解析范围扩展到上一层父元素（<p>标签），也要面对相同样式的问题。所以，解析这样的树形结构要比之前的方法二复杂得多。

如果你就是想用BeautifulSoup来完成数据提取，我的建议是先用浏览器的Developer Tools功能好好观察一下页面的结构，比如在Chrome浏览器里，你可以将鼠标移到你想要查看的元素上——也就是这个项目中的日期和时间部分——点击鼠标右键，然后选择Inspect Element。这样就会打开一个Developer Tools面板，你可以从完整的文档树中找到对应数据的所在位置。每个HTML元素左侧的箭头符号表明该元素是否还有子节点。这个时候，你可以决定如何以编程的方式来定位树形结构中的目标元素，并进一步做好辨别不同节点的计划。这项任务的细节无法在此尽述，所以我会把它留作练习。

5

5.6 小结

在这一章中，我们学习了一些行之有效的数据分离技术。在厨房煲汤时，我们总会使用一种过滤工具把不想要的骨头和蔬菜残叶过滤掉，只让香浓可口的汤水流入容器中。这种思路同样适用于在数据科学厨房中提取网页中的数据。我们需要设计一种清洗计划，只保留需要的数据，去除不需要的HTML内容。

学习过程中，我们掌握了两种从HTML页面中提取数据的心智模型，它们分别是行分隔模型和树形结构模型。然后又深入学习了三种解析HTML页面的有效方法：正则表达式，BeautifulSoup，还有基于Chrome浏览器的Scraper工具。最后，我们将所学的知识融合到一起，通过从电子邮件和HTML页面收集与提取数据，共同完成了一个完整的项目。

像电子邮件和HTML这样的文本数据并不难清洗，那二进制文件又如何呢？在接下来的一章中，我们将学习如何从难度更大的PDF文件中抽取数据。

第 6 章

清洗PDF文件中的数据

在上一章中我们得知，要想从大量数据中把我们需要的数据分离出来，可以使用多种不同的方法。其实我们可以把数据清洗过程想象成制作鸡汤，我们的目的是滤掉鸡骨，保留剩下的汤水。但是，如果我们所需要的数据和不需要的数据不能轻易地分开，那又该怎么办呢？

假设我们有一杯陈年美酒，里面有些许杂质。乍一看，可能确实看不到杯中有沉积物存在。只有经过了一段时间，等到沉积物全部落到杯底，我们才能倒出纯净清澈的美酒。在这种情况下，使用简易的过滤器是不可能把酒和沉积物分开的，我们得用一个特殊的工具。

在这一章，我们将采用各种实验手段来提取隐藏在PDF里的好东西。其中涉及的主题包含：

- PDF的用途是什么，以及为什么很难从PDF文件中抽取数据
- 怎么复制和粘贴PDF文件内容，以及如何应对不能复制和粘贴的情况
- 怎么从目标PDF文件中保存需要的页面
- 怎么利用Python的pdfMiner包中的工具从PDF文件中抽取文本和数字
- 怎么利用基于浏览器的Java应用程序Tabula从PDF文件中抽取表格数据
- 怎么使用完整的付费版Adobe Acrobat抽取表格数据

6.1 为什么 PDF 文件很难清洗

以可移植文档格式（PDF）存储的文件，比本书到目前为止描述过的文本文件都要复杂一些。PDF文件是二进制格式的，最初由Adobe Systems公司发明，后来发展成为一种开放标准，许多应用程序都可依靠此标准来创建它们自己的PDF文档。PDF文件的目的在于提供一种能够独立于专门的布局软件而进行文字和图像查看的方法。

20世纪90年代初期是桌面应用鼎盛的时期，每种图形设计软件包都有其专属的一套文件格式，而且这些软件包个个都价格不菲。当时，为了查看Word、Pagemaker或Quark文档，你必须使用创建这些文档的软件来打开它们。对于早期的网络环境来说，情况可能更为严重，因为当时没有太多的HTML技术可以用来创建丰富的布局，但即便如此人们还是热衷于彼此分享文件。这就

自然地促使PDF成为一种厂商中立的布局格式。Adobe把Acrobat Reader软件免费开放，供人们下载，这使得PDF格式广为流行。

> 这里有一个关于初期Acrobat Reader的故事。在Google搜索引擎中输入单词“click here”，结果集中的第一条记录就是Adobe’s Acrobat PDF Reader download website，而且这一状况一直持续了好多年。这是因为许多网站在发布PDF文件的时候，常常配上一句“To view this file you must have Acrobat Reader installed. Click here to download it.”（要想查看此文件，必须安装 Acrobat Reader。点击此处下载。）。因为Google的搜索算法使用链接上的文字来获取与关键字有关的网站，所以关键字“click here”对应的结果就是Adobe Acrobat的下载网站。

PDF至今还被用于创建厂商和应用中立的文件，这些文件格式有着纯文本文件无可比拟的多样性布局。当我们以不同版本的Microsoft Word查看同一份文档时，有时候内容看起来会不一致，这可能体现在文档内嵌的表格、样式、图片、表单或字体上。而导致这些问题的因素有很多，比如操作系统的不同或是安装的Word版本不同。就算是软件包或版本都兼容，也有可能产生一些细微的差别。而PDF则能帮助解决这些问题。

我们马上就会明白为什么PDF要比纯文本文件难处理，因为它本身是二进制的，内嵌字体、图片等内容。所以我们之前用过的大部分数据清洗工具，如文本编辑器和各种命令行工具（`less`），在碰到PDF文件的时候基本上都无用武之地了。但幸运的是，我们还有别的方法从PDF文件中提取数据。

6.2 简单方案——复制

假设你想给自己倒一杯甘醇可口的红酒，可一不小心洒到地板上了。你的第一反应可能会觉得很沮丧，因为整条地毯都要更换。但在开始收拾之前，你最好像有经验的老酒保那样处理一下现场，这需要准备足够的苏打水和一条湿抹布。在这一节，我们也要为即将开启的文件翻新项目做些准备工作。这些方案可能会有效，也可能无效，但不管怎样还是值得一试的。

6.2.1 我们的实验文件

还是让我们用一个真实的PDF文件来做数据清洗练习吧。为了不让实验太过容易，我们得找一个比较复杂的文件。假设我们需要从皮尤研究中心的网站上一个名为“Is College Worth It?”的文件中拉取数据。该PDF文件发布于2011年，长达159页，其中含有大量数据表格，显示了衡量在美国上大学是否是一项值得的投资的各种方式。我们要做的就是找到一种从这些表格中快速提取数据的方法，以便在这些数据上做一些额外的统计工作。比如，下面就是报告中使用到的表格之一。

Q.23/Q.24, **BASED ON TOTAL**[31]

Which of these would you say is the most important role colleges and universities play in society?

All		4-year Private	4-year Public	2-year Private/Public	For profit
38	Prepare students to be productive members of the workforce	23	28	47	63
27	Prepare young people to be responsible citizens	48	31	14	8
21	Ensure that all qualified students have equal access to a college education	16	24	25	20
4	Contribute to the economic development of their region or locality	2	5	8	2
4	Conduct research to help solve medical, scientific, social, and other national problems	5	8	3	1
2	Provide continuing education for adults of all ages	2	1	1	3
3	None is “very important”/No answer	4	4	3	2

这个表格非常复杂。虽然只有六列八行，但其中有几行数据占用了两行文本，而且只有五个列有标题行。

> 完整的报表可以在皮尤研究中心的网站找到：http://www.pewsocialtrends.org/2011/05/15/is-college-worth-it/，而我们要用的PDF文件位于：http://www.pewsocialtrends.org/files/2011/05/higher-ed-report.pdf。

6.2.2　第一步：把我们需要的数据复制出来

这个例子中要使用的实验数据可以从PDF文件的149页（文档中标注的页码是143）找到。如果用PDF查看工具打开文件，比如Mac OS X的Preview，然后选择表格中的数据，我们会发现一些奇怪的事情。举个例子来说，即使我们没有选中页码（143），它也会自动出现在选择范围中。这并不利于我们的实验，但不要管它了，还是继续吧。使用组合快捷键Command-C或是菜单Edit | Copy把数据复制出来。

143

Q.23/Q.24, **BASED ON TOTAL**[31]

Which of these would you say is the most important role colleges and universities play in society?

All		4-year Private	4-year Public	2-year Private/Public	For profit
38	Prepare students to be productive members of the workforce	23	28	47	63
27	Prepare young people to be responsible citizens	48	31	14	8
21	Ensure that all qualified students have equal access to a college education	16	24	25	20
4	Contribute to the economic development of their region or locality	2	5	8	2
4	Conduct research to help solve medical, scientific, social, and other national problems	5	8	3	1
2	Provide continuing education for adults of all ages	2	1	1	3
3	None is "very important"/No answer	4	4	3	2

PDF文件内容在Preview中被选中的样子

6.2.3 第二步：把复制出来的数据粘贴到文本编辑器中

下面的截图显示的是复制出来的文本被粘贴到文本编辑器Text Wrangler中的样子。

```
38 Prepare students to be productive
members of the workforce
27 Prepare young people to be responsible
citizens
21 Ensure that all qualified students have
4-year 4-year 2-year Private Public Private/Public
23 28 47 48 31 14 16 24 25
For profit
63 8 20
143
     equal access to a college education
4 Contribute to the economic development 2582
of their region or locality
4 Conduct research to help solve medical, 5831
scientific, social, and other national
problems
2 Provide continuing education for adults 2113
of all ages
3 None is “very important”/No answer 4432
```

很明显，数据在复制粘贴之后看起来毫无顺序。而且当中包含了页码信息，数字信息也不是像原先那样垂直分布，而是全都变成水平分布，每列的标题位置也错乱了。虽然有些数字还是排列在一起，比如最后一行中包含了数字4,4,3,2，但在粘贴之后它们变成一个数字4432了。如果采

用手工清洗的方法，这得花上相当长的一段时间，还不如把表格里的数据重新录入一次。就这个PDF文件来说，我们的结论就是得采取一些更强的清洗手段。

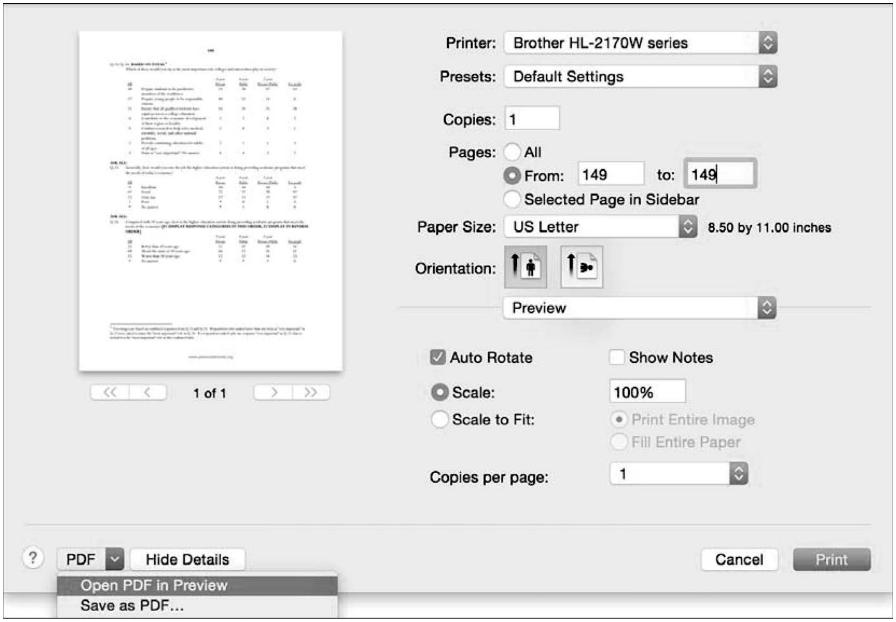
> 此时我们应该注意的是，这个PDF文件的其他内容还是比较好清洗的。比如序言，即位于文件第三页的纯文字部分，只需使用前面提到的数据复制技术就可以把数据提取出来。其实难点只是表格里的数据而已。在确定一种数据提取方法之前，你应该对PDF文件中的所有内容都实验一下，包括文本数据和表格数据。

6.2.4　第三步：轻量级文件

刚才尝试的复制粘贴方法并不奏效，所以我们得找出更多可以使用的方法。其实我们不需要把这个PDF中的159页数据都抽取出来，只需要识别出我们需要的，然后把它们保存到另一个文件中就行了。

这项工作可以在Mac OS X上的Preview里完成。通过菜单File | Print...开启打印对话框。在Pages区域，我们需要键入实际想要复制的页码范围。针对这个实验来说，我们只需要第149页的内容，所以我们只需像下面截图那样在From:和to:文本框里输入149就可以了。

然后从底部的PDF下拉框中选择Open PDF in Preview。之后你会在新的窗口中看到一个只有单个页面的PDF文件。从这里我们可以把它保存为一个新的文件并对它重新命名，比如report149.pdf。



6.3　第二种技术——pdfMiner

文件已经变小了，现在我们可以试试用编程的方案来抽取文本数据，看看情况是不是会有转机。pdfMiner是Python的一个包，内置了一些可以操作PDF文件的工具。在这个实验中我们会用到一个叫作`pdf2txt`的命令行工具，通过这个工具我们可以从PDF文档中抽取文本数据。也许这个方法可以帮我们正确地拿到表格里的数字。

6.3.1　第一步：安装 pdfMiner

启动Canopy Python环境。从Canopy Terminal Window中运行下面的命令：

```
pip install pdfminer
```

这一步将安装pdfMiner包以及与之关联的命令行工具。

> 关于pdfMiner以及它自带的`pdf2txt`和`dumpPDF`工具的全部文档信息，可以从http://www.unixuser.org/~euske/python/pdfminer/获得。

6.3.2　第二步：从 PDF 文件中提取文本

我们可以使用命令行工具`pdf2txt.py`从PDF文件中提取文本数据。为此，需要使用Canopy Terminal并转向文件所在目录。命令的基本格式为`pdf2txt.py <filename>`。如果你要处理的是一个包含有多个页面的大文件（或者你没有像上面那样把文件变小），那么你可以使用`pdf2txt.py -p149 <filename>`来告诉程序只处理第149页中的内容。

与前面复制与粘贴的例子一样，我们不单单会尝试149页的表格，也会实验第3页的序言。为了从第3页提取文本数据，我们运行下面的命令就可以了：

```
pdf2txt.py -p3 pewReport.pdf
```

命令运行完成之后，抽取出来的皮尤研究中心的序言内容就会出现在命令行窗口：

```
ch6 — Canopy Terminal — bash — 80×24
(Canopy 64bit) flossmole2:ch6 megan$ pdf2txt.py -p3 HigherEdReport.pdf


EMBARGOED FOR RELEASE AT 8 P.M. EDT ON SUNDAY, MAY 15
Preface

Sharply rising college costs, enrollments and student debt loads have touched of
f a debate
about the role of higher education in the 21st Century.

This Pew Research Center report attempts to inform that debate. It is based on t
wo surveys—
one of the American public; the other of college presidents—that explore attitud
es about the
cost, value, quality, mission and payoff of a college education.  The survey of
college presidents
was done in association with the Chronicle of Higher Education.

As is the case with all Center reports, our research is not designed to promote
any cause,
ideology or policy proposal. Our only goal is to inform the public on important
topics that
shape their lives and their society.
```

如果想要把文本保存成一个名为pewPreface.txt的文件，只需为命令添加一个重定向操作：

```
pdf2txt.py -p3 pewReport.pdf > pewPreface.txt
```

那149页里的表格数据该怎么处理呢？如果用`pdf2txt`会发生些什么呢？我们可以试着运行一下下面的命令：

```
pdf2txt.py pewReport149.pdf
```

结果看起来比复制粘贴确实好一点点，但也没有我们期望的那么好。实际的数据输出结果如下图所示。列的标题和数据全都混杂在一起，而且不同列的数据也都是胡乱排列的。

```
ch6 — Canopy Terminal — bash — 80×24
4-year
Private

23

4-year
Public
28

48

16

2

5

2

4

31

24
```

我们不得不承认这个实验的表格数据抽取是失败的，虽然pdfMiner在逐行的文本抽取方面很在行。

> 不同的工具在数据抽取上会有不同的效果，主要取决于PDF文件内容的格式特性。

看来我们选的这个PDF例子确实有难度，但不要灰心。我们可以继续探索下一个工具，看看能有什么样的效果。

6.4　第三种技术——Tabula

Tabula是一个基于Java的程序，也可以用来提取PDF文件中的表格数据。现在我们要做的就是下载Tabula并把它应用到149页中的表格上。

6.4.1　第一步：下载 Tabula

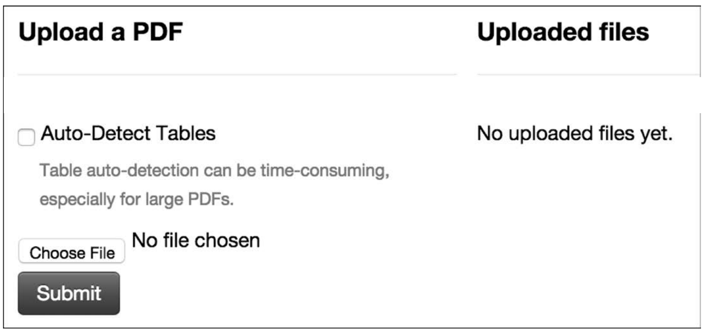
Tabula可以从http://tabula.technology/上下载。这个网站同时还提供了一些简单的下载指南。

> 在10.10.1版本的Mac OS X系统上，我下载了Java 6来运行Tabula。这个过程比较简单，只需要按照屏幕上的提示来操作就可以。

6.4.2　第二步：运行 Tabula

从下载下来的.zip归档文件中启动Tabula。在Mac系统中，Tabula应用程序文件的名字是Tabula.app。如果你愿意的话，还可以把它复制到Applications文件夹。

Tabula启动的时候，它会在你的默认浏览器中新开一个标签页或是窗口，并把地址设置成http://127.0.0.1:8080/。页面初始部分的截图如下。



警告说自动发现表格功能会花费一段时间，这一点确实是真的。这是因为只是处理一个包含三个表格的单页面文件perResearch149.pdf，表格自动发现就耗费了整整两分钟时间，并且还在最后给出了一个PDF文件格式不正确的错误消息。

6.4.3　第三步：用 Tabula 提取数据

在Tabula读取文件后，我们就可以用它提取表格数据了。用你的鼠标指针选择希望提取的表格。我的做法是围绕第一个表格画了一个框。

Tabula一共花费了30秒读完这个表格，结果显示如下。



		4-year	4-year	2-year	
All		Private	Public	Private/Public	For profit
38	Prepare students to be productive	23	28	47	63
	members of the workforce				
27	Prepare young people to be responsible	48	31	14	8
	citizens				
21	Ensure that all qualified students have	16	24	25	20
	equal access to a college education				
4	Contribute to the economic development	2	5	8	2
	of their region or locality				
4	Conduct research to help solve medical,	5	8	3	1
	scientific, social, and other national				
	problems				
2	Provide continuing education for adults	2	1	1	3

与之前的数据复制粘贴和`pdf2txt`方法相比，这次数据看起来更有样儿了。不过如果你对Tabula读出来的表格数据还是不满意的话，可以反复执行这个操作，清除选择，然后重新绘制选择区域。

6.4.4　第四步：数据复制

我们可以使用Tabula里的Download Data按钮把文件保存成合适的文件格式，比如CSV或TSV。从前面几章的工作结果中我们已经知道，这些格式都可以根据需要在电子表格或文本编辑器中进行清洗。说做就做，赶快为下一步做好准备吧。

6.4.5　第五步：进一步清洗

用Excel或文本编辑器打开CSV文件，然后仔细观察其中的格式和内容。由于之前我们在抽

取PDF数据时失败过很多次，所以这一次也很容易轻言放弃。但是回顾一下以往学过的清洗知识，你可能马上就会猜到，数据还有进一步清洗的余地。基于在前几章中学过的技术，我们可以执行以下步骤来轻松地完成数据清洗任务。

(1) 我们可以把所有占用两行的文本单元格合并成一个单元格。例如，在B列中，有许多跨行的文本。Prepare students to be productive和members of the workforce应该作为一条完整的语句放在同一个单元格里。相似的情况也发生在第一行和第二行的标题中（4-year和Private应该放在一个单元格里）。要在Excel中把这些内容清理掉，需要在B列和C列之间创建一个新列。然后使用函数`concatenate()`把B3:B4、B5:B6等单元格结合起来。再使用选择性粘贴（Paste-Special）把联结起来的值添加到新列中去。最后就可以删除不再需要的那两列。接着以同样的操作清洗第一行和第二行的标题内容。

(2) 删除行与行之间的空白行。

这些步骤都完成后，数据看起来应该是下面的样子：

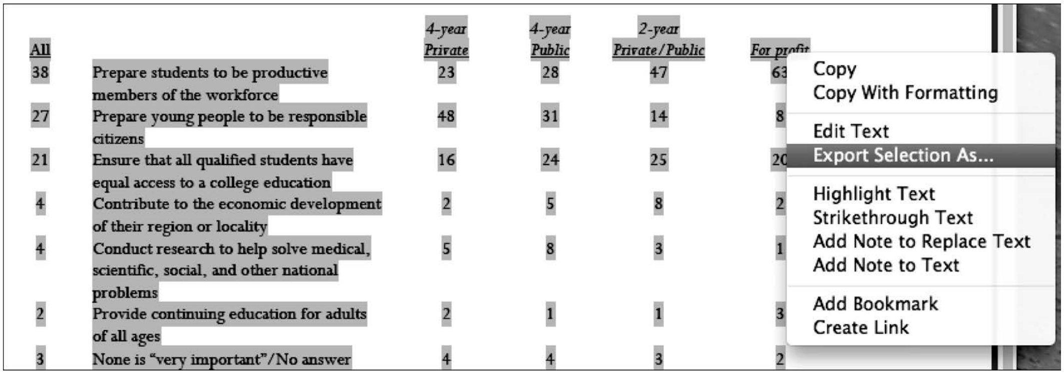
	A	B	C	D	E	F
1	All		4-year Private	4-year Public	2-year Private/Public	For profit
2	38	Prepare students to be productive members of the workforce	23	28	47	63
3	27	Prepare young people to be responsible citizens	48	31	14	8
4	21	Ensure that all qualified students have equal access to a college education	16	24	25	20
5	4	Contribute to the economic development of their region or locality	2	5	8	2
6	4	Conduct research to help solve medical, scientific, social, and other national problems	5	8	3	1
7	2	Provide continuing education for adults of all ages	2	1	1	3
8	3	None is "very important" /No answer	4	4	3	2

与剪切和粘贴数据，或是使用命令行工具相比，使用Tabula看起来需要我们多做许多工作。而事实也的确如此，除非你的PDF文件格式规矩得不得了。记住，每种专项工具都有它们存在的理由，但我们仅应该在必要的时候才使用它们。在执行数据清洗操作的时候，应该首先选用简单的解决方案，然后再根据需要尝试难度稍为大一些的工具。

6.5 所有尝试都失败之后——第四种技术

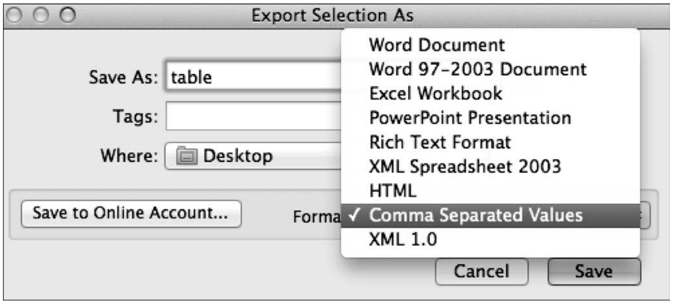
Adobe Systems出售一种付费的商业版Acrobat软件，除了我们之前提到的PDF文件读取功能外，还包括一些别的功能。完整版的Acrobat可以帮我们创建复杂的PDF文件，并以各种方式对既存的文件进行操作。其中与我们实验有关的一项功能就是Acrobat的Export Selection As...。

要想使用这个功能，先要启动Acrobat并使用File Open对话框打开PDF文件。文件打开之后，转到要输出的数据表格所在的页面。下面的截图中演示的是如何选择PDF第149页的数据。用鼠标选中数据，然后点击鼠标右键并选择Export Selection As...。



All		4-year Private	4-year Public	2-year Private/Public	For profit
38	Prepare students to be productive members of the workforce	23	28	47	63
27	Prepare young people to be responsible citizens	48	31	14	8
21	Ensure that all qualified students have equal access to a college education	16	24	25	20
4	Contribute to the economic development of their region or locality	2	5	8	2
4	Conduct research to help solve medical, scientific, social, and other national problems	5	8	3	1
2	Provide continuing education for adults of all ages	2	1	1	3
3	None is “very important”/No answer	4	4	3	2

这时，Acrobat会问你想以什么方式输出数据。CSV格式便是选项之一。如果你不想在文本编辑器里编辑文件，那Excel工作簿（`.xlsx`）也是一个不错的选择。不过Excel也可以用来打开CSV文件，所以我决定使用这个格式，这样就可以最大化地利用Excel和文本编辑器两者的优势。



选择好文件格式后，程序会提示我们设置文件的名字和存储位置。当用文本编辑器或Excel打开新的结果文件时，我们可以看到，这些数据看起来与前一节表格里的数据几乎一模一样。下面就是CSV在Excel中打开的样子。



	A	B	C	D	E	F
1	All		4-year Privat	4-year Public	2-year Privat	For profit
2	38	Prepare stud	23	28	47	63
3		members of the workforce				
4	27	Prepare your	48	31	14	8
5		citizens				
6	21	Ensure that a	16	24	25	20
7		equal access to a college education				
8	4	Contribute tc	2	5	8	2
9		of their region or locality				
10	4	Conduct rese	5	8	3	1
11		scientific, social, and other national				
12		problems				
13	2	Provide cont	2	1	1	3
14		of all ages				
15	3	None is Òver	4	4	3	2

这时，我们可以使用曾经在清洗Tabula数据时用过的清洗流程，把B2:B3这样的跨行单元格合并成一个单元格，然后再删除多余的空行。

6.6 小结

这一章的目的是学习如何从PDF文件中导出数据。就像酒中的沉积物那样，PDF文件里的数据初看起来也是很难分离的。但与倒酒的被动过程不同，在从PDF中分离出数据的过程中，我们经历了许多尝试与失败。我们学会了四种清洗PDF数据的方法：复制与粘贴、pdfMiner、Tabula和Acrobat导出。每种工具方法都有自己的优势与缺陷。

- 复制粘贴成本低，工作量小，但对于复杂的表格不起作用。
- pdfMiner/Pdf2txt是免费的命令行工具，并可以应用在自动化处理中。针对大量数据的情况也可以使用。但与复制粘贴一样，遇到某些类型的表格数据就无计可施了。
- Tabula需要我们做一些额外的设置工作，并且该产品仍处于开发阶段，有时候还会给出一些奇怪的警告信息。与其他方案相比速度也相对慢一些。但它的输出结果非常整洁，即便是在面对复杂的表格时也能应对自如。
- Acrobat的数据输出效果与Tabula类似，几乎不需要做任何设置，人工成本很低。但我们得支付一定的费用才可以使用它。

到最后，我们拿到了一份干净的数据集，可以直接用于数据分析或者是长期存储。

在接下来的一章中，我们将要把重点放在以关系型数据管理系统（RDBMS）为载体的长期存储的数据上。我们将学习如何清洗以这种方式存储的数据，以及修正一些常见的不规则数据。

第 7 章

RDBMS清洗技术

家用冰箱全都自带置物架，并且大多会有一两个用来装蔬菜的抽屉。不过如果你去过家庭分类商店，或是与专业的分类专家交流过，就会发现有各种各样的存储选项可供我们选择，包括蛋托、芝士盒、苏打水枪、酒架、剩菜标记系统，以及各种尺寸、可嵌套叠加、以不同颜色进行区分的储物罐等。但这些真的都是我们需要的吗？要想回答这个问题，你得先问问自己，常用的食物是不是每次都能轻而易举地找到？食物是不是占用了过多的空间？剩下的食材是不是都明确地打上了标记以方便确认？如果答案全都是否定的话，那么分类专家会告诉你，容器和标签贴纸可以帮助你优化存储、减少浪费，让你的生活更加轻松。

对于关系型数据库管理系统（Relational Database Management System，RDBMS）来说，情况也是相同的。作为经典的长期数据存储解决方案，RDBMS可以说是现代数据科学工具箱的标配。但与此同时，由于缺少对数据库细节内容的考虑，我们常常会对存放其中的数据感到一丝愧意。在这一章里，我们会学习如何设计RDBMS，而并非像一两个置物架加几个抽屉那么简单。我们要学的新技术可以让RDBMS的存储得到优化，减少浪费，使我们的工作更加轻松。具体来说，我们将：

- 学会找出RDBMS中的异常数据
- 学会使用不同的策略来清洗不同类型的问题数据
- 学会因地制宜地创建新的数据表来存放清洗后的数据，包括创建子表和查找表
- 学会编写文档，以便时刻掌握数据的变化

7.1 准备

为了能够顺利地完成本章中的例子，我们需要准备一份称为Sentiment140的流行数据集。建立这个数据集是为了帮助我们了解Twitter消息中的正面和负面情绪。我们不是真的要做情绪分析，只是想要把它导入关系型数据库，然后用来做一下数据清洗的练习而已。

要想使用Sentiment140数据集，你得安装好MySQL服务器，就跟之前Enron例子中做的一样。

7.2　第一步：下载并检查 Sentiment140

我们要使用的Sentiment140数据版本是一个由两个文件组成的数据集，这个数据集源自Sentiment140项目，可以从网站http://help.sentiment140.com/for-students获取相关的项目信息。要使用的ZIP文件是由斯坦福大学的几个研究生创建的，当中包含了推文以及根据推文内容所创建的正负面情绪数据（分为0、2、4三个等级）。由于这个文件是公开的，所以一些网站就把原版的Sentiment140文件添加到许多大型推文数据集中。但在这一章我们只需使用原版的Sentiment140文本文件就足够了，这个文件可以从前面提到的网站链接中获取，或者是直接从http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip下载。

下载ZIP文件，解压，然后用文本编辑器查看解压出来的两个CSV文件。你马上会注意到，其中一个文件的内容要比另一个文件多得多，不过它们的字段个数是却是相同的。数据是使用逗号分隔的，每个字段都采用双引号进行了封闭处理。对于每个字段的描述信息，可以从上一段for-students链接所对应的页面中找到。

7.3　第二步：清洗要导入的数据

对于我们的目的——学习如何清洗数据，只要把少量数据加载到MySQL数据表中就足够了。既然我们学习所需的所有内容用一个小一点的文件就能搞定，那么testdata.manual.2009.06.14.csv正合适。

在观察数据的时候我们会发现，有些地方的数据可能不会顺利地导入到MySQL数据库。问题之一就出现在文件的第28行：

```
"4","46","Thu May 14 02:58:07 UTC 2009","""booz allen""",
```

在关键词`booz`的前面和关键词`allen`的后面分别有三个连续的双引号`"""`，看到了吧？同样的双引号问题在第41行的歌曲标题`P.Y.T`前后也出现了：

```
"4","131","Sun May 17 15:05:03 UTC 2009","Danny
Gokey","VickyTigger","I'm listening to ""P.Y.T"" by Danny Gokey..."
```

这个问题的根源是，MySQL导入程序是通过引号来界定字段文本内容的。所以多余的引号会让MySQL错误地认为这一行含有的字段个数多于实际的个数。

其实这个问题也不难修复，只要使用文本编辑器的查找与替换功能就能解决，做法就是把所有的`"""`替换成一个`"`（双引号），把所有的`""`替换成`'`（单引号）。

7

> 在这个清洗练习中，完全删除""也不会有什么负面影响。只需在替换""的时候将替换内容置成空就可以了。但是如果你坚持保留原有的推文形式，那在替换的时候使用单引号或转义后的双引号\"会更为稳妥。

把清洗后的文件内容保存到一个新命名的文件中，比如cleanedTestData.csv。现在可以往MySQL中导入新文件了。

7.4　第三步：把数据导入 MySQL

想要把清洗后的数据文件导入MySQL，需要使用3.1.3节“往MySQL中导入电子表格数据”中介绍的方法。

(1) 通过命令行将当前工作目录切换到在第二步创建的新文件目录。这个新文件就是我们将要导入到MySQL的数据文件。

(2) 之后启动MySQL客户端并连接到数据库服务器：

```
user@machine:~/sentiment140$ mysql -hlocalhost -umsquire -p
Enter password:
```

(3) 输入你的登录密码，成功登录之后使用下面的命令在MySQL中创建数据库，该数据库是为了下一步创建数据表而准备的：

```
mysql> CREATE DATABASE sentiment140;
mysql> USE sentiment140;
```

(4) 接下来需要创建存放数据的数据表。每个字段的数据类型和长度都应该与我们持有的数据相匹配。有些字段需要采用变长字符类型，并且需要指定具体长度。也许此时我们并不知道该为每个字段指定什么样的长度，不过我们可以使用清洗工具来获取一个合适的长度范围。

(5) 如果在Excel（也可以使用Google Spreadsheets）中打开CSV文件，可以运行一些简单的函数来找到文本字段的最大长度。比如使用函数`len()`可以返回文本字符串中的字符个数，使用函数`max()`可以找出一个指定范围内的最大数字。在打开CSV文件的前提下，可以应用这些函数来找出在MySQL中需要以变长字符为类型的字段的长度。

下面的截图显示的是使用函数解决这个问题的具体操作方法。其中函数`len()`应用在G列上，H列上的函数`max()`应用在G列。

	A	B	C	D	E	F	G	H
1	4	3	Mon May 11 03:17:40 UT(	kindle2	tpryan	@stellargirl I looooooooow	=LEN(F1)	=MAX(G1:G498)
2	4	4	Mon May 11 03:18:03 UT(	kindle2	vcu451	Reading my kindle2... Lov	=LEN(F2)	
3	4	5	Mon May 11 03:18:54 UT(	kindle2	chadfu	Ok, first assesment of the	=LEN(F3)	

G列和H列分别演示了如何在Excel中获取文本列的长度以及长度的最大值。

(6) 要想更快地计算出每个字段的最大长度，还可以使用Excel的快捷键。利用下面的数组公式可以快速地将文本类型单元格的长度计算和最大值计算结合起来——只要确保你在输入嵌套函数之后用组合键Ctrl+Shift+Enter代替Enter就可以了：

```
=max(len(f1:f498))
```

这个嵌套函数可以应用在任何文本列上以取得其对应的文本最大长度，并且这个函数只占用一个单元格而不需要逐个计算单元格的长度。

在运行这些函数之后，我们可以看到推文的最大长度是144个字符。

7.4.1　发现并清洗异常数据

你也许会好奇，为什么这个数据集中的推文会有144个字符，Twitter明明规定每个推文的最大长度为140个字符。其实这是因为在Sentiment140数据集中，有时候字符&会被转义为等价的HTML编码&amp，但有时不会。类似的情况也会发生在其他字符上，比如符号<会被转义为<，符号>会被转义为>。所以，对于少数较长的推文来说，这些字符的存在可以轻松地突破140个字符的限制。因为我们知道这些HTML编码字符并非出自推文的原始作者，而且这种转义时而发生时而不会发生，所以我们称这些数据为*异常数据*。

要清洗这些异常数据，有两个方案可以选择。第一个方案是先把数据导入数据库，然后在数据库端进行数据清洗。第二个方案是在Excel或文本编辑器中进行清洗。为了区分这两种技术之间的差别，我们会把两个方案都演示一遍。首先，使用电子表格或文本编辑器中的查找与替换功能对下面表格中的字符进行转换。我们可以先把CSV文件中的数据导入Excel，看看到底有多少清洗工作可以在这里完成。

HTML编码	替换字符	替换个数	用于查找个数的Excel函数
<	<	6	=COUNTIF(F1:F498,"*&lt*")
>	>	5	=COUNTIF(F1:F498,"*&gt*")
&	&	24	=COUNTIF(F1:F498,"*&amp*")

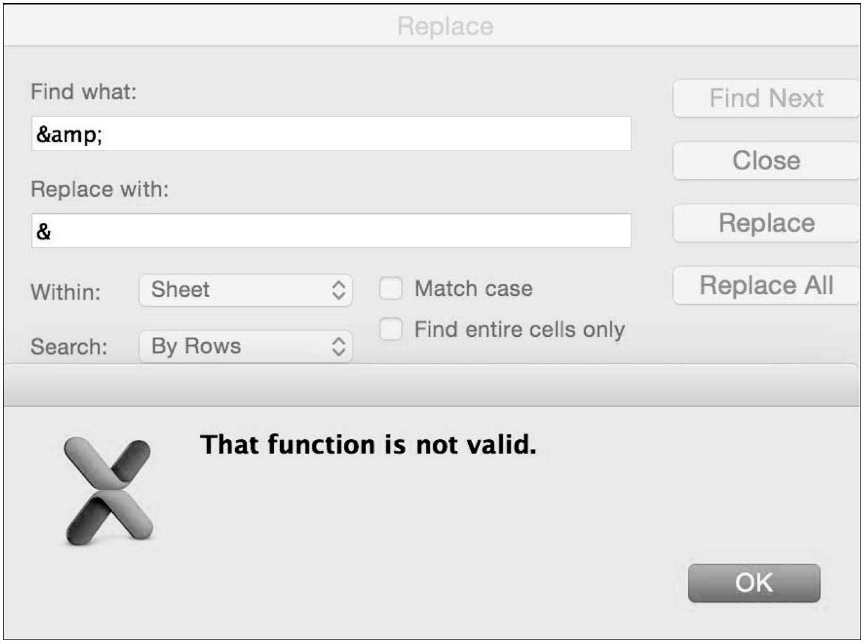
前两个字符的替换可以通过Excel中的*查找与替换*轻松完成。HTML编码<和>全会被替换掉。请看下面的文本示例：

```
I'm listening to 'P.Y.T' by Danny Gokey &lt;3 &lt;3 &lt;3 Aww, he's so amazing. I &lt;3
  him so much :)
```

上面的文本被修改成：

```
I'm listening to 'P.Y.T' by Danny Gokey <3 <3 <3 Aww, he's so amazing. I <3 him so much :)
```

但是，当我们尝试使用Excel把&替换成&时，有可能会遇到下面的错误：



在有些操作系统上，或是某些版本的Excel中，在使用字符&作为替换内容时会有问题。如果在操作时恰好碰到这样的错误，就需要采用一些不同的方法。

- 可以使用搜索引擎找出针对这个错误的Excel解决方案。
- 可以把CSV文本数据放到文本编辑器里，执行查找与替换操作。
- 可以先继续推进工作，把数据全都放到数据库里，即便其中含有奇怪的`&`字符也没关系，因为我们会在数据库里进行清洗。

正常情况下，如果数据可以在数据库以外的地方清洗，我就不会把脏数据放到数据库里。但这一章要讲的就是如何在数据库内部做数据清洗，所以还是让我们把清洗一半的数据导入到数据库中，然后再进一步清洗数据表中的`&`。

7.4.2　创建自己的数据表

要想把清洗一半的数据转移到数据库中，需要先编写创建数据表用的`CREATE`语句，然后在数据库中运行。`CREATE`语句如下：

```
mysql> CREATE TABLE sentiment140 (
    ->   polarity enum('0','2','4') DEFAULT NULL,
    ->   id int(11) PRIMARY KEY,
    ->   date_of_tweet varchar(28) DEFAULT NULL,
    ->   query_phrase varchar(10) DEFAULT NULL,
    ->   user varchar(10) DEFAULT NULL,
```

```
    ->   tweet_text varchar(144) DEFAULT NULL
    -> ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

> 这条语句会使用简单快捷的MyISAM引擎，因为我们并不需要使用InnoDB的行级锁或事务这样的功能。关于MyISAM和InnoDB之间更多的区别，请参考存储引擎的讨论话题：http://stackoverflow.com/questions/20148/myisam-versus-innodb。

也许你已经注意到，上面的代码中仍旧使用144作为字段`tweet_text`的长度。这是因为我们还没有清洗`&`编码的缘故。但就算是这样也不会有什么问题，因为我们知道变长字符类型的字段不会使用额外的存储空间，除非真的需要。这就是它们被称作变长字符、可变字符或可变字段的原因。如果这些额外的长度确实困扰你了，那稍后你可以把这些字段的长度修改成140个字符。

接下来我们可以在数据文件的所在位置使用MySQL命令行程序运行下面的数据导入语句：

```
mysql> LOAD DATA LOCAL INFILE 'cleanedTestData.csv'
    ->   INTO TABLE sentiment140
    ->   FIELDS TERMINATED BY ',' ENCLOSED BY '"' ESCAPED BY '\'
    ->   (polarity, id, date_of_tweet, query_phrase, user,
         tweet_text);
```

这个命令会把之前清洗过的CSV文件加载到新创建的数据表中。执行完成后，系统会像下面那样显示一条成功的消息，这代表498行记录全都加载到数据表中了：

```
Query OK, 498 rows affected (0.00 sec)
Records: 498  Deleted: 0  Skipped: 0  Warnings: 0
```

> 如果你能够访问像phpMyAdmin这样基于浏览器的接口，或者是像MySQL Workbench或MySQL版本的Toad这样的桌面应用程序，那就可以直接在这些工具中更加方便地执行以上所有的SQL命令，而无需在命令行中手动输入，比如在phpMyAdmin中，你可以从Import页签上传CSV文件。但请确保文件已经按照7.3节的步骤清洗过，否则文件中的很多字段都会出错。这个错误主要还是由引号引起的。

7.5　第四步：清洗&字符

在上一个步骤中，我们决定推迟对`&`字符的清洗，因为在做替换操作时Excel给出了奇怪的错误。既然我们已经完成第三步把数据存放在MySQL里了，现在就可以使用`UPDATE`语句和`replace()`函数进行后续的数据清洗工作。使用下面的SQL查询语句会将所有的`&`都替换成`&`：

```
UPDATE sentiment140 SET tweet_text = replace(tweet_text,'&amp;', '&');
```

函数replace()的工作原理与Excel或文本编辑器中的查找与替换功能完全一样。我们可以看到ID为594的推文，已经由原来的#at&t is complete fail变为#at&t is complete fail。

7.6　第五步：清洗其他未知字符

当我们仔细检查字段tweet_text的时候，可能会发现一些奇怪的推文内容，比如ID为613和2086的记录：

```
613, Talk is Cheap: Bing that, I?ll stick with Google
2086, Stanford University?s Facebook Profile
```

其中字符?应该引起我们的注意。与之前碰到的HTML编码字符一样，它很有可能也是在字符集转换时产生的。这种情况一般都是因为原始推文中含有high-ASCII或Unicode编码的撇号（有时候也称为智能引号），但这些数据在向低位字符集转换的时候，比如纯ASCII，特殊风格的撇号就会被简单地转换成?。

要不要在数据中保留带有?的字符，这完全取决于处理数据的目的。假如我们要做字数统计或是文本挖掘，那就必须把I?ll转换成I'll，把University?s转换成University's。如果我们认为这样做很有必要的话，那接下来的工作就是从推文数据中找出问题数据，然后设计一套转换方案，把问号重新转回单引号。这个操作需要一定的技巧，绝对不可以简单地把字段tweet_text中的每一个问号直接替换成单引号字符，因为推文中还有一些问号应该保持它们原本的样貌。

可以通过运行一些SQL查询语句并配合正则表达式来确定有问题的数据。有一点需要注意的是，我们要查找的问号字符所处的位置比较特殊，它们的后面往往会紧紧跟随着另一个字母。所以，在下面的语句里，我们需要使用MySQL的REGEXP功能。通过这条命令可以大致找到存在问题的问号位置：

```
SELECT id, tweet_text
FROM sentiment140
WHERE tweet_text
REGEXP '\\?[[:alpha:]]+';
```

SQL中的正则表达式会匹配后面跟随着一个或多个字母的问号。查询一共产生六行结果，其中四行包含有问题的问号，而另外两行则是*不当匹配*。不当匹配的数据虽然满足匹配样式，但实际上是不应该被修改的。这两处不当匹配的推文ID分别是234和2204，其中包含的问号其实是URL的一部分。而推文139、224、613和2086才是*正确匹配*的数据，这就意味着其中确实包含了异常数据并且应该纠正。下面的phpMyAdmin截图中显示的是所有的查询结果：

id	tweet_text
139	?Obama Administration Must Stop Bonuses to AIG Ponzi Schemers ... http://bit.ly/2CUlg
224	Life?s a bitch? and so is Dick Cheney. #p2 #bipart #tlot #tcot #hhrs #GOP #DNC http://is.gd/DjyQ
234	jQuery UI 1.6 Book Review - http://cfbloggers.org/?c=30631
613	Talk is Cheap: Bing that, I?ll stick with Google. http://bit.ly/XC3C8
2086	Stanford University?s Facebook Profile is One of the Most Popular Official University Pages - http://tinyurl.com/p5b3fl
2204	Learning jQuery 1.3 Book Review - http://cfbloggers.org/?c=30629

经过仔细观察，ID为139的推文看起来还是有点古怪。它的问号在单词Obama之前，看起来像是新闻文章中对名字的引用，但字符串末端并没有结束用的引号（缺少配对的引号）。这有可能是别的什么字符吗？实际上这也可能是不当匹配，或者至少不足以成为正确匹配以进行修复。再进一步观察，224也在本不该出现问号的地方出现了奇怪的问号字符。

如果我们要使用`replace()`函数把有问题的问号替换成单引号，就得编写一个正则表达式，要求是只匹配有问题的数据，绝不可以错误匹配到没有问题的数据。但是，由于这个数据集的规模比较小，且只有四处是真正有问题的地方（如果连139这条记录也排除了，那就只剩下三处），我们完全可以手工完成清洗工作。这种方式对于处理像224这样的问题数据来说，真是再也适合不过了。

既然我们只剩下三行要处理的问题数据，那直接写出三条`UPDATE`命令远比构建一个完美的正则表达式快得多。下面的SQL查询语句可以用来修复推文224（只针对第一个问号）、613和2086：

```
UPDATE sentiment140 SET tweet_text = 'Life''s a bitch? and so is
  Dick Cheney. #p2 #bipart #tlot #tcot #hhrs #GOP #DNC
  http://is.gd/DjyQ' WHERE id = 224;

UPDATE sentiment140 SET tweet_text = 'Talk is Cheap: Bing that,
  I''ll stick with Google. http://bit.ly/XC3C8' WHERE id = 613;

UPDATE sentiment140 SET tweet_text = 'Stanford University''s
  Facebook Profile is One of the Most Popular Official University
  Pages - http://tinyurl.com/p5b3fl' WHERE id = 2086;
```

> 请注意，我们必须在更新语句中对单引号进行转义。在MySQL中，转义字符要么是反斜线，要么是单引号本身。上面的例子就是使用单引号作为转义字符。

7.7　第六步：清洗日期

如果我们仔细查看字段`date_of_tweet`中的内容，很容易就能发现它是一个变长字符类型的`varchar(30)`字段。这有什么问题吗？假设我们想按时间从早到晚的顺序对推文数据进行排序。我们不可能只通过简单的`ORDER BY`子句得到正确的日期排序，因为我们只能得到以字母顺序排序的数据。所有的Friday都会排在Monday的前面，所有的May都会排在June后面。我们可以

利用下面的SQL查询来验证这一说法：

```
SELECT id, date_of_tweet
FROM sentiment140
ORDER BY date_of_tweet;
```

前几行顺序还是对的，但到了28行左右问题就出现了：

```
2018  Fri May 15 06:45:54 UTC 2009
2544  Mon Jun 08 00:01:27 UTC 2009
...
3  Mon May 11 03:17:40 UTC 2009
```

May 11不应该排在May 15或是Jun 8的后面。要修复这个问题，我们需要再创建一个新的日期时间字段来存放清洗后的字符串数据。我们在2.3.2节中学过，MySQL可以很好地处理date、time或是datetime数据。日期时间组合类型的数据格式可以是：YYYY-MM-DD HH:MM:SS。但在目前的date_of_tweet字段中，数据并不是这样。

MySQL提供了大量的内置函数，可以帮助我们把格式混乱的日期字符串格式化成我们希望的样子。通过这种方式，我们可以在MySQL里对日期和时间数据执行一些数学操作，比如找出两个日期或时间的间隔，或者是按照对应的日期和时间来进行排序。

为了把字符串转换成MySQL可以接受的日期时间类型，需要执行以下操作。

(1) 修改原来的数据表并加入一个新的字段，这样做的目的是为了存放新生成的日期时间信息。我们可以把这个新字段命名为date_of_tweet_new或date_clean，或者是其他与原来的date_of_tweet字段明显不同的新名字。下面是完成这项任务需要用到的SQL语句：

```
ALTER TABLE sentiment140
ADD date_clean DATETIME NULL
AFTER date_of_tweet;
```

(2) 针对每一行数据执行更新语句，在这个操作中我们会把旧的日期字符串格式化成格式更加规范的日期时间类型，并把新得到的数据放到刚刚创建的date_clean字段中。下面是完成这项任务需要用到的SQL语句：

```
UPDATE sentiment140
SET date_clean = str_to_date(date_of_tweet,
   '%a %b %d %H:%i:%s UTC %Y');
```

此时我们已经拥有一个存满干净数据的新字段了。之前的旧字段date_of_tweet无法正确地排列日期。那新字段是不是能够派上用场呢？我们可以用下面的语句来测试一下：

```
SELECT id, date_of_tweet
FROM sentiment140
ORDER BY date_clean;
```

从结果中我们可以看出排序效果十分理想，5月11号的记录排在第一行，且不存在顺序不对的数据。

那我们是不是该把旧的日期字段删除了呢？这个问题还是由你自己来决定吧。如果你担心会有什么差错，或者是出于某些特殊的原因还需要这些原始数据，那就把它们保留下来。但是如果你不想保留这些旧数据了，那就可以删除这个字段：

```
ALTER TABLE sentiment140
DROP date_of_tweet;
```

另外，你还可以选择为数据表Sentiment140创建一个副本，然后在副本中保留原始字段作为备份使用。

7.8 第七步：分离用户提及、标签和 URL

这份数据中现在还有一个问题，就是有许多信息潜藏在字段`tweet_text`中，比方说一个用户发布的推文想要引起另一个用户的注意，就会在后者的用户名前面加上一个@符号。这就是Twitter上的提及（mention）。或许我们可以统计一下每个用户被提及的次数，或者是计算出他们与某个关键词同时出现的次数。另外还有一部分标签（hashtag）信息隐藏在推文中，比如在ID为2165的推文中，人们用分别使用标签`#jobs`和`#sittercity`来讨论工作和保姆的事情。

这条推文中同时还包含了一个Twitter以外的URL信息。接下来我们要做的就是分别提取用户提及、标签和URL信息，然后把它们保存到数据库中。

这项任务与我们之前的日期清洗任务比较相似，但是有一个重要的区别。在日期的案例中，我们只有一个需要纠正的数据，所以只要添加一个新的字段存放清洗后的数据即可。但是对于用户提及、标签和URL来说，字段`tweet_text`中可能会同时包含零个或多个这样的数据，比如我们前面所说的推文（ID为2165），它就包含了两个标签，与此相似的还有下面的推文（ID为2223）：


```
HTML 5 Demos! Lots of great stuff to come! Yes, I'm excited. :)
  http://htmlfive.appspot.com #io2009 #googleio
```

在这条推文中，没有包含用户提及信息，但是包含了一个URL和两个标签信息。而推文13078中则有三个用户提及，但它并不包含标签和URL：

```
Monday already. Iran may implode. Kitchen is a disaster. @annagoss
  seems happy. @sebulous had a nice weekend and @goldpanda is
  great. whoop.
```

我们需要修改数据库的结构来存放这些新的信息——标签、URL和用户提及。另外要记住的是，被处理的推文可能会包含许多这样的数据。

7.8.1　创建一些新的数据表

如果遵循关系型数据库的设计理念，我们应该避免创建用于存放多个属性值的字段。例如，如果一条推文中含有三个标签的话，不要把它们都放到一个字段中。这种设计原则使得我们无法简单地照搬之前在日期清洗中用过的ALTER方案。

取而代之的方案是创建三张全新的数据表：sentiment140_mentions、sentiment140_urls和sentiment140_hashtags。每张新表都有一个主键ID字段，除此之外只包含两个字段：第一个是tweet_id，用来建立新表与原始表sentiment140之间的关系，另一个则是抽取出来的标签、用户提及或URL信息。下面是三个用于创建表的CREATE语句：

```
CREATE TABLE IF NOT EXISTS sentiment140_mentions (
  id int(11) NOT NULL AUTO_INCREMENT,
  tweet_id int(11) NOT NULL,
  mention varchar(144) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS sentiment140_hashtags (
  id int(11) NOT NULL AUTO_INCREMENT,
  tweet_id int(11) NOT NULL,
  hashtag varchar(144) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS sentiment140_urls (
  id int(11) NOT NULL AUTO_INCREMENT,
  tweet_id int(11) NOT NULL,
  url varchar(144) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

> 这些表没有使用外键来引用存放在原始表sentiment140中的数据。如果你想添加外键约束，当然也是可以的。但对于数据集清洗练习来说，真的没有这个必要。

既然数据表已经创建好了，接下来就该从tweet_text字段提取数据完成相应的数据填充工作了。我们会分别讲述每一种数据抽取过程，先从用户提及开始。

7.8.2　提取用户提及

在设计用户提及的提取逻辑之前，先让我们回顾一下推文中用户提及的样式：

- 用户提及必须以@符号开头
- 用户提及是紧随@符号之后的那个单词

- ❑ 如果@符号后面有空格，那它就不是用户提及
- ❑ 用户提及内部是不允许出现空格的
- ❑ 我们得小心电子邮件地址中的@符号

根据上面的样式规则，我们可以先构造一些有效的用户提及样式：

- ❑ @foo
- ❑ @foobar1
- ❑ @_1foobar_

接下来还可以构建一些无效的用户提及例子：

- ❑ @ foo（@符号后面的空格会导致用户提及样式匹配无效）
- ❑ foo@bar.com（bar.com是无法识别的）
- ❑ @foo bar（只有@foo才会被识别出来）
- ❑ @foo.bar（只有@foo才会被识别出来）

> 在这个例子中，我们假定不需要区分@mention和.@mention这两种形式，后者有时也称为点提及（推特的一种语法）。这些推文都是在@符号前面放置一个句点。这种语法的目的是把推文消息内容发送给当前用户的所有关注者。

要是完全采用SQL实现这组规则，实现起来一定非常复杂，所以还是先写一小段简单的脚本，并配合一些正则表达式来完成清洗工作吧。连接数据库的脚本不论用什么语言编写都是可以的，Python或PHP等随你选择。因为我们在第2章中用过PHP连接数据库，所以这里还使用它。下面这段脚本会先连接到数据库，然后在tweet_text字段中搜索用户提及，随后将找到的用户提及放到新的sentiment140_mentions表中：


```
<?php
// 连接数据库
$dbc = mysqli_connect('localhost', 'username', 'password',
  'sentiment140')
    or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");

// 提取推文数据
$select_query = "SELECT id, tweet_text FROM sentiment140";
$select_result = mysqli_query($dbc, $select_query);

// 如果查询失败则终止处理
if (!$select_result)
    die ("SELECT failed! [$select_query]" .  mysqli_error());

// 提取用户提及信息
$mentions = array();
```

```
while($row = mysqli_fetch_array($select_result))
{
    if (preg_match_all(
        "/(?<!\pL)@(\pL+)/iu",
        $row["tweet_text"],
        $mentions
    ))
    {
        foreach ($mentions[0] as $name)
        {
            $insert_query = "INSERT into sentiment140_mentions (id,
tweet_id, mention) VALUES (NULL," . $row["id"] . ",'$name')";
            echo "<br />$insert_query";
            $insert_result = mysqli_query($dbc, $insert_query);
            // 如果查询失败则终止处理
            if (!$insert_result)
                die ("INSERT failed! [$insert_query]" .
                mysqli_error());
        }
    }
}
?>
```

运行完脚本之后，我们会发现在sentiment140表所包含的489条推文中，一共有124个非重复的用户提及。另外还有一点要指出的是，这段脚本能够很好地处理用户名中的Unicode字符，不过在这个测试数据集中并没有这样的数据。我们可以马上向数据表sentiment140末尾插入一条测试数据来验证我们的说法：

```
INSERT INTO sentiment140 (id, tweet_text) VALUES(99999, "This is a @ тест");
```

现在让我们重新运行一次脚本。你会立马看到数据表sentiment140_mentions中加入了一条新数据，其中用户提及是采用Unicode编码的@**тест**。在下一节中，我们将构建一个类似的脚本来提取标签信息。

7.8.3 提取标签

标签也有自己的定义规则，而且与用户提及有一些区别。下面列举了部分规则，我们可以根据它们来判断所给的数据是不是标签。

- 标签以符号#开始。
- 标签单词会紧紧跟随符号#。
- 标签中可以包含下划线，但不可以包含空格或其他标点符号。

这段提取标签的PHP代码几乎和我们前面写的用户提及提取程序完全一样，只是代码中间的正则表达式有所不同。我们需要把变量从$mentions改成$hastags，然后再把正则表达式调整成下面的样式：

```
if (preg_match_all(
        "/(#\pL+)/iu",
        $row["tweet_text"],
        $hashtags
    ))
```

这段正则表达式的含义是，在不区分大小写的情况下匹配Unicode字符。之后我们需要修改INSERT语句，使用正确的数据表名字和字段名字：

```
$insert_query = "INSERT INTO sentiment140_hashtags (id, tweet_id,
  hashtag) VALUES (NULL," . $row["id"] . ",'$name')";
```

在成功运行这段脚本之后，我们会看到一共有54个标签记录加入到数据表sentiment140_hashtags中。其中多条推文包含多个标签，甚至比包含多个用户提及的推文还要多。我们以推文174和224为例，这两条记录都包含了多个嵌套标签。

接下来我们还是使用同样的脚本框架，稍作修改之后就可以用来提取URL信息了。

7.8.4 提取 URL

从文本中提取URL信息可以像查找以http://或https://开头的字符串那样简单。当然，这个过程也可能会因文本字符串包含的URL类型不同而变得复杂。举个例子来说，有些字符串可能包含file://这样的URL，也可能包含torrent链接，比如磁力链接，或者是别的不太常见的链接。但就我们这个项目的Twitter数据来讲，分析起来还是比较容易的，因为数据集中的URL全都以HTTP开头。所以，我们可以稍微偷一下懒，只需简单设计一个正则表达式，专门用来提取http://或https://后面的字符串就足够了。正则表达式的样式如下：

```
if (preg_match_all(
        "!https?://\S+!",
        $row["tweet_text"],
        $urls
    ))
```


然而，如果我们试着用一下搜索引擎，很容易发现一些让人感到惊讶的通用URL匹配样式，利用它们可以搞定更多的复杂链接样式。使用这些样式的意义在于，即使在未来的某一天数据发生了变化，我们还是可以使用同样的程序来处理这些新的复杂状况。

从网站http://daringfireball.net/2010/07/improved_regex_for_matching_urls上可以找到一份文档齐全的URL样式匹配程序。下面的代码演示了该如何修改PHP代码来利用这个样式提取数据集Sentiment140中的URL：

```
<?php
// 连接数据库
$dbc = mysqli_connect('localhost', 'username', 'password', 'sentiment140')
    or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");
```

```
// 提取推文数据
$select_query = "SELECT id, tweet_text FROM sentiment140";
$select_result = mysqli_query($dbc, $select_query);

// 如果查询失败则终止处理
if (!$select_result)
    die ("SELECT failed! [$select_query]" .  mysqli_error());

// 提取URL
$urls = array();
$pattern  = '#\b(([\w-]+://?|www[.])[^\s()<>]+(?:\([\w\d]+\)|([^[:punct:]\s]|/)))#';

while($row = mysqli_fetch_array($select_result))
{
    echo "<br/>working on tweet id: " . $row["id"];
    if (preg_match_all(
        $pattern,
        $row["tweet_text"],
        $urls
    ))
    {
        foreach ($urls[0] as $name)
        {
           echo "<br/>----url: ".$name;
           $insert_query = "INSERT into sentiment140_urls (id,
           tweet_id, url)
                        VALUES (NULL," . $row["id"] . ",'$name')";
                   echo "<br />$insert_query";
                   $insert_result = mysqli_query($dbc, $insert_query);
                   // 如果查询失败则终止处理
                   if (!$insert_result)
                       die ("INSERT failed! [$insert_query]" .
                       mysqli_error());
        }
    }
}
?>
```

这个程序几乎和我们前面写的用户提及提取程序完全一样，只是有两处例外。第一处是存放正则表达式的`$pattern`变量，现在这个样式又长又复杂。第二处是我们对`INSERT`命令做了少量修改，做法同标签抽取部分的代码一样。

这里所使用的正则表达式样式的完整解释可以从它的原始网站上找到，但简单一点讲，就是可以利用这个样式来匹配任何URL协议，比如http://或file://，同时它还会匹配有效的域名和带有层级的目录关系（目录/文件）。从原始网站中我们还可以找到更多的样式和它们对应的测试数据，有的演示的是可以明确匹配的样式，而有的却是相反的示例。

7.9　第八步：清洗查询表

在7.8节中，我们创建了几个新表来存放抽取出来的标签、用户提及和URL信息，并可以通

过其中的id字段找到原始表中对应的数据。根据数据库标准化原则，这些新表分别在推文数据与用户提及、推文数据与标签、推文数据与URL之间，创建了一系列一对多的关系。在这一步，我们将继续对数据表进行优化，提升它们的性能与效率。

我们现在重点关注的字段是query_phrase。从这个字段中的内容可以看出，很多数据都是重复的。这些数据明显是当初用来定位与筛选数据集中推文信息的搜索短语。在sentiment140表里存放的498条推文中，有多少这样重复的查询短语呢？我们可以使用下面的SQL来检查一下：

```
SELECT count(DISTINCT query_phrase)
FROM sentiment140;
```

查询结果显示不重复的查询短语只有80条，但实际上它们被反复用在了498条记录上。

对于一个只有498条记录的表来说，这好像也没什么，但如果我们现在面对的是一张数据量超大的表，比如有上亿条数据，有两件事就不得不考虑了。第一件就是反复重复的字符串占用了本无需使用的数据库空间，第二件就是数据检索起来会很慢。

要解决这个问题，需要创建一张*查询表*来另外保存这些查询数据。在新表中，每个查询短语只允许出现一次，并且还要为它们创建ID编码。之后要做的是修改原始表中的数据，用新的数字编码来替代现在使用的字符串值。完整的操作过程如下。

(1) 创建一个新的查找表：

```
CREATE TABLE sentiment140_queries (
  query_id int(11) NOT NULL AUTO_INCREMENT,
  query_phrase varchar(25) NOT NULL,
  PRIMARY KEY (query_id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```

(2) 用不重复的查询短语填充查找表，并自动为字段query_id生成一个编号：

```
INSERT INTO sentiment140_queries (query_phrase)
SELECT DISTINCT query_phrase FROM sentiment140;
```

(3) 在原来的表中创建一个新的字段来存放查询短语编号：

```
ALTER TABLE sentiment140
ADD query_id INT NOT NULL AFTER query_phrase;
```

(4) 为了防止下一步有什么意外发生，先对表sentiment140进行备份。无论什么时候执行UPDATE操作，备份总是好的。要想创建表sentiment140的副本，可以使用phpMyAdmin这样的工具轻松地完成表复制的工作（该功能位于Operations页签中）。另外还有一种方法就是重新创建一个新表，然后从原始表导入数据，具体的SQL如下：

```
CREATE TABLE sentiment140_backup(
  polarity int(1) DEFAULT NULL,
  id int(5)NOT NULL,
  date_of_tweet varchar(30) CHARACTER SET utf8 DEFAULT NULL
```

7

```
  ,
  date_clean datetime DEFAULT NULL COMMENT 'holds clean,
  formatted date_of_tweet',
  query_id int(11) NOT  NULL,
  user varchar(25) CHARACTER SET utf8 DEFAULT NULL,
  tweet_text varchar(144) CHARACTER SET utf8 DEFAULT NULL ,
  PRIMARY KEY (id)) ENGINE=MyISAM DEFAULT CHARSET=utf8;

SET SQL_MODE='NO_AUTO_VALUE_ON_ZERO';
INSERT INTO sentiment140_backup SELECT * FROM sentiment140;
```

(5) 为新字段填写正确的编号。为此，需要以文本字段为连接条件把这两张表关联起来，然后再从查询表中找出正确的编码值，把找到的编码插入到表sentiment140中。在下面的查询中，每个表都有对应的别名，分别是s和sq：

```
UPDATE sentiment140 s
INNER JOIN sentiment140_queries sq
ON s.query_phrase = sq.query_phrase
SET s.query_id = sq.query_id;
```

(6) 从表sentiment140中删除旧的query_phrase字段：

```
ALTER TABLE sentiment140
DROP query_phrase;
```

至此，我们已经可以利用创建好的短语列表来高效地实现排序操作了。下面就是以字母顺序排序的查询语句：

```
SELECT query_phrase
FROM sentiment140_queries
ORDER BY 1;
```

另外，还可以通过把两张表关联起来找出指定短语（baseball）所对应的推文信息：

```
SELECT s.id, s.tweet_text, sq.query_phrase
FROM sentiment140 s
INNER JOIN sentiment140_queries sq
  ON s.query_id = sq.query_id
WHERE sq.query_phrase = 'baseball';
```

到这里，我们已经完成对表sentiment140的清洗工作，并且还拥有了四张含有干净数据的表，分别是标签、用户提及、URL和查询短语。字段tweet_text和date_clean也已经清洗干净，并且查询短语也都重新规划到查询表。

7.10　第九步：记录操作步骤

在九步清洗过程中，我们使用了多种编程语言和工具，只要其中某一步稍有差池，就得把这一步的操作全部重新再来一次。如果我们必须跟其他人解释曾经做过些什么，也很难回想起每一

个操作细节与动机。

要想记录下我们一路犯下的错误，保存一份关于清洗步骤的详细日志是非常必要的。而且这个日志至少也应该以清洗步骤为顺序，记录下每一个操作中所涉及的以下几个方面内容。

- 每条SQL语句。
- 每个Excel函数和文本编辑器程序，如果有必要可以加入演示截图。
- 每个脚本程序。
- 关于历史操作的笔记和注释。

还有一种比较好的思路就是在每个阶段都创建一个备份表，例如，我们在对数据表`sentiment140`执行`UPDATE`操作之前就创建了一张备份表，在创建新字段`date_clean`之后也做了备份。备份实现起来比较容易，而且不论什么时候，只要不需要了就可以马上删除。

7.11 小结

在这一章里，我们通过一份名为Sentiment140的简单推文数据集，学习了如何清洗和操作保存在关系型数据库管理系统中的数据。前期的一些基本清洗工作都是在Excel中完成的，然后数据从CSV文件被转移到了数据库。之后的清洗操作就全部集中在RDBMS内部了。之后，学习了如何把字符串转换成日期格式，又演练了从推文中提取三种数据的过程，最后把提取出来的干净数据放到新的数据表。此后我们又学习了如何创建查询表来存放那些低效存储的数据，并通过数字来提高更新效率、优化查询速度。但是，由于我们执行了许多步骤，里面有可能存在潜在的错误或是理解有误的地方，所以，在最后一个步骤中，我们总结了一些与数据清洗相关的文档编写策略。

在下一章中，我们将把视角从数据清洗转移到数据分享。我们将学习一些创建数据集的最佳实践，与他人分享几乎无需进行清洗就能直接使用的数据。

第 8 章 数据分享的最佳实践

到目前为止，我们已经学会了许多清洗与组织数据集的方法。也许是时候考虑把干净的数据拿出来与大家分享了。所以，这一章的目标就是把朋友们邀请到数据科学的厨房里，向他们展示我们的成果。数据共享意味着要把数据提供给其他人、其他团队，或者只是未来的你。那应该用什么方式来对数据进行包装？应该以什么方式来告知人们你有哪些清洗过的数据？如何确保你的辛勤劳动成果都归功于你自己？

在这一章里我们将学习：

- 如何呈现并打包你清洗好的数据
- 如何为数据提供一份清晰明了的文档
- 如何通过许可协议来保护并扩展你的工作成果
- 如何查找并评估数据推广的方法

在正式开始这一章的内容之前，有一件事应该事先说清楚，我们只应清洗与分享我们有权处置的数据。这也许本来就是不言自明的事情，但提醒一下还是有必要的。在本章中，我们假定你要清洗的数据以及后续要分享的数据，都是你有权处理的。如果你对此有疑问，请阅读8.3节，以确保你和你的用户都能严格遵守相关的守则。

8.1 准备干净的数据包

在这一节中，我们会探讨在发布数据包之前需要回答的几个重要问题。

你想让人们通过什么样的途径来访问你的数据呢？如果是在一个数据库环境，你想让用户登录到数据库并运行SQL命令访问这些数据吗？还是说你觉得创建一份可以下载的普通文本文件更方便一些？再或者是创建一个数据访问用的API？你的数据量到底有多大？是否需要对不同的数据采用不同的访问级别呢？

从技术层面看，数据的分享方式是很重要的。一般来讲，我们应该先从简单的方案入手，然后在必要的时候根据具体需求做出更为精细的发布计划。下面是一些有关数据发布的可选方案，

其顺序是按照复杂程度进行组织的。当然，计划做得越完备周详，收益效果就会越明显。

- **压缩的纯文本数据**——这可以说是一种风险极低的发布方法。我们在第2章中曾学过，纯文本格式的文件可以被压缩成体积很小的文件。用途广泛的CSV或JSON就属于这样的文本格式，而且它们很容易被转换成其他格式。但在具体操作中还需考虑以下几点内容。
    - 你想让用户通过什么样的方式下载文件呢？直接通过网页上的链接下载既简单又方便，但可惜的是它不能对文件访问进行授权控制，无法只允许提供了正确的用户名和密码的用户访问文件。如果这项功能对你来说比较重要的话，那么你可以考虑改换另一种文件发布方法了，比如使用需要提供用户名和密码才能访问的FTP服务器，或者是在网络服务器加上访问控制。
    - 还有，你的文件会有多大呢？期望的访问流量又是多少？在不额外支付费用的前提下，你的主机提供商能为你提供多少流量？
- **压缩的SQL文件**——发布SQL文件可以让你的用户在他们自己的系统上重新创建数据库结构和数据。但在具体操作中还需考虑以下几点内容。
    - 用户运行的数据库系统可能和你的不太一样，所以他们还得额外做数据清洗。如果事实真是这样的话，提供纯文本文件的效率可能会更高一些。
    - 你的数据库系统设置可能与用户的不同。如果是这样的话，你就必须事先对这些自定义项目进行解释说明。
    - 你需要提早为数据集的未来变化做好应对计划，比如是否只提供`UPDATE`语句更新数据变化，或者是提供足够多的`CREATE`和`INSERT`语句重新创建整个数据库。
- **在线数据库访问**——直接为用户提供数据库访问是一种可以让用户访问底层数据的好办法。但在具体操作中还需考虑以下几点内容。
    - 如果提供在线访问功能，你就必须为每一位用户提供一套独立的用户名和密码，这就意味着你可以追踪用户的操作行为。
    - 由于用户都是可以识别的，你需要提供一种方法来回应他们遇到的各种问题，比如授权信息丢失和系统的使用方法。
    - 使用通用的用户名和密码可不是什么好主意，除非你为数据库构建了一套安全的前端接口，并做好了一些基本的预防措施，比如限制用户能够执行的查询语句个数，以及查询所占用的时长。如果一个`OUTER JOIN`查询引入了数据量在几个T级别的数据表，很有可能造成数据库宕机，从而影响其他用户的使用。
    - 你是否想让用户通过编写程序的方式来访问数据，比如使用ODBC或JDBC这样的中间件？如果是的话，请把访问权限做到计划之中，并做好相应的服务器配置。

8

- API——设计一个允许用户访问数据的应用编程接口（API）可以让最终用户编写他们自己的数据访问程序，并以可预测的数据形式接收结果集。API的优势在于它能以一种已知的、受限的方式通过互联网访问数据，同时用户也不需要解析数据文件或纠结于数据格式的转换。但在具体操作中还需考虑以下几点内容。
  - 构建一个好的API，成本比之前列举的所有选择都要高。但是，如果你有大量的客户群，同时负责支持的员工又比较有限，那么从长远角度来看构建API确实会节省不少资金投入。
  - 与其他方法相比，使用API需要用户掌握更多的技术知识。你需要准备好足够的文档和必要的演示程序。
  - 你需要做好授权与安全计划来跟踪数据访问者以及他们的行为。如果你计划为数据访问设置不同的级别，例如，将数据按照不同的层次进行划分，就应该事先为用户提供一种可以提升他们访问级别的机制。
  - 与一般的数据库访问一样，用户可能会造成API的过度使用或滥用。你需要提早做好计划并采取预防措施来定位并移除那些含有恶意或粗心的用户，因为这些用户很有可能通过有意或无意的滥用导致服务无法使用。

选择哪种发布方法与你的预算（既包括资金预算，也包括时间预算）和用户的期望有很大关系。我能给你的最好建议就是我一直以来都在遵循的开源软件格言——早发布和常更新。这对我来说还是挺管用的，因为我的社区用户群体比较小，而且预算有限，也没有太多的闲暇时间去做那些尚不知有效还是无效的打包计划。

警句箴言——使用 GitHub 发布数据

GitHub是一个基于云的文件仓库，它的设计目的是为了帮助软件开发者在软件开发中进行协作，并且可以存放可供他人下载的代码。这种方式在当下是很流行的，其中存放的项目已经超过了1600万。正是出于这个原因，与我交流过的许多数据科学家都会建议在GitHub上存放数据。

虽然GitHub因它的普遍性和易用性而著称，但在存储非代码类型的数据时还是有一些局限性的，我们应该注意那些可能对数据造成影响的规则与条款。这些规则和条款在GitHub的帮助指南中都有提到，其访问地址为https://help.github.com/articles/what-is-my-disk-quota，我们在这里将其归纳为以下两点。

- 首先，GitHub是对源代码控制系统Git的包装，Git系统不是专门用来存放SQL文件的。帮助文档中已经说明："像Git这样的版本控制系统不适合管理大型的SQL文件。"我本人对"不适合"的含义并不是十分理解，但我能够确信的是，我不希望用户不高兴。

- 其次，GitHub对文件大小有着严格的限制。它明确规定了每个项目（仓储）的存储上限为1 GB，每个独立文件的存储上限为100 MB。好在我发布的大部分数据文件的体积都小于这个限制，但由于每年都要发布许多以时间为顺序的文件，所以我不得不建立多个仓储。在这种限制规定之下，每当要发布新文件的时候，我都得先做一下检查，看看是不是有超过文件大小限制的文件。这件事很快就让我感到头疼。

简而言之，GitHub本身推荐使用网络主机的方式来发布文件，特别是在这些文件体积较大或是它们专门面向数据库的时候。如果你决定把文件放在GitHub上，请一定要小心对待文件中的用户授权信息。这包括数据库系统的用户名和密码，Twitter账号认证用的密钥和加密信息，或者是与其他个人隐私相关的内容。因为GitHub的核心就是Git仓库，所以你犯下的所有错误会被永远地保留下来，除非是仓库被删除。如果你发现不小心把个人信息发布到GitHub上了，那就应该立即重新设置所有账户的认证信息，并重新创建加密文件和密码。

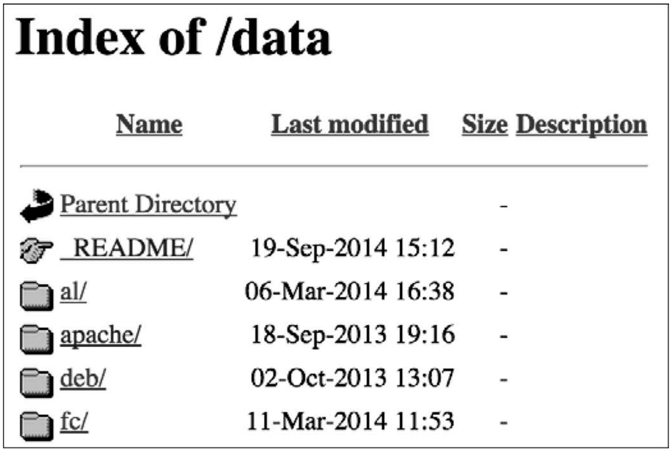
8.2　为数据编写文档

在用户取得数据访问权限时，理想的情况是他们事先已经清楚自己会拿到什么样的数据。虽然为数据编写文档这项任务对你来说可能是后续的额外工作，但对于用户来说这是非常重要的事情，因为他们不可能像你一样对数据的每一处细节都十分了解，也无法得知这些数据都经过什么样的处理。在这一节中，我们会对数据包中的内容逐一进行检查，以确保它们比较容易理解。

8.2.1　README 文件

README文件在计算机上有一段较长的历史。它就是一个普通的文本文件，随着软件包一同发布，有时会和其他文件一起放在一个目录中，其目的是让用户先阅读README文件，然后再使用其余的文件。README文件会告知用户有关软件包的一些重要信息，比如作者是谁，出于什么目的编写了这个软件，安装指南，已知的问题，还有其他的基本使用指南。

如果你正准备构建一些数据包，比如压缩一些文本文件或是SQL文件，那么最好是在压缩之前就把这个README文件准备好。如果你正在创建为数据文件而准备的网站或在线目录，在一个显眼的地方添加README文件效果会非常明显。下面截图中显示的就是我曾发布过的一个名为FLOSSmole的网站目录。当时我添加了一个README目录，并将所有希望用户阅读的文件放了进去。另外我还在目录名字的前面添加了一个下划线，这样就可以让它在按照字母排序显示的时候列在第一位上。



包含README文件的目录显示在最顶端

在README.txt文件内部，我针对里面包含的文件为用户编写了概要说明和详细指南。下面是其中一个README文件的内容示例：

```
README for http://flossdata.syr.edu/data directory

What is this place?
This is a repository of flat files or data "dumps", from the FLOSSmole
project.

What is FLOSSmole?
Since 2004, FLOSSmole aims to:
    --freely provide data about free, libre, and open source software
(FLOSS) projects in multiple formats for anyone to download;
    --integrate donated data & scripts from other research teams;
    --provide a community for researchers to discuss public data about
FLOSS development.
FLOSSmole contains: Several terabytes (TB) of data covering the period
2004-now, and growing with data sets from nearly 10,000 web-based
collection operations, and growing each month. This includes data for
millions of open source projects and their developers.

If you use FLOSSmole data, please cite it accordingly:
Howison, J., Conklin, M., & Crowston, K. (2006). FLOSSmole: A
collaborative repository for FLOSS research data and analyses.
International Journal of Information Technology and Web Engineering,
1(3), 17–26.

What is included on this site?
Flat files, date and time-stamped, from various software forges &
projects. We have a lot of other data in our database that is not
available here in flat files. For example, IRC logs and email from
various projects. For those, see the following:

1. Direct database access. Please use this link for direct access to
```

```
our MySQL database: http://flossmole.org/content/direct-db-access-
flossmole-collection-available

2. FLOSSmole web site. Includes updates, visualizations, and examples.
http://flossmole.org/
```

这个README文件对整个目录中的所有文件做了说明，你也可以为每一个文件或目录创建一份README文件。具体怎么做由你说了算。

8.2.2 文件头

另外还有一种与用户高效沟通的方式，就是在每个文件的头部放置一些关于文件格式和用途的解释性文字，这种做法尤其适用于文本文件和SQL文件。一种常见的实施方案是在文件的头部为每一行说明文字添加注释字符前缀，比如#或//。

常常包含在文件头部的信息有：

- 文件名和其所在的包名
- 文件作者或是所有协作者的名字，以及他们的组织和所在地
- 文件发布日期
- 文件版本号，及其早期版本的所在位置
- 文件的用途
- 原始数据来源，数据前前后后经过哪些处理
- 文件格式与组织方法，比如列举出各个字段和它们对应的含义
- 文件使用条款和许可协议

下面演示的文件头部信息取自我在之前一个数据项目中发布的一个TSV文件。其中我对数据内容做了解释说明，并给出了每个字段的解析方法。我同时规定了数据引用和分享的相关条款。我们会在稍后对许可协议和分享做进一步讨论。

```
# Author: Squire, M. & Gazda, R.
# License: Open Database License 1.0
# This data 2012LTinsultsLKML.tsv.txt is made available under the
# Open Database License: http://opendatacommons.org/licenses/
# odbl/1.0/.
#
# filename: 2012LTinsultsLKML.tsv.txt
# explanation: This data set is part of a larger group of data
# sets described in the paper below, and hosted on the
# FLOSSmole.org site. Contains insults gleaned from messages sent
# to the LKML mailing list by Linus Torvalds during the year 2012
#
# explanation of fields:
# date: this is the date the original email was sent
# markmail permalink: this is a permalink to the email on markmail
```

```
# (for easy reading)
# type: this is our code for what type of insult this is
# mail excerpt: this is the fragment of the email containing the
# insult(s). Ellipses (...) have been added where necessary.
#
# Please cite the paper and FLOSSmole as follows:
#
# Squire, M. & Gazda, R. (2015). FLOSS as a source for profanity
# and insults: Collecting the data. In Proceedings of 48th
# Hawai'i International Conference on System Sciences (HICSS-48).
# IEEE. Hawaii, USA. 5290-5298
#
# Howison, J., Conklin, M., & Crowston, K. (2006). FLOSSmole: A
# collaborative repository for FLOSS research data and analyses.
# International Journal of Information Technology and Web
# Engineering, 1(3), 17-26.
```

如果你预测用户会定期收集你的数据文件的话，那就应该在文件头部使用一致的注释字符。在前一个例子中，我使用的字符是#。这样做的原因是用户可能会使用程序自动下载并解析数据，可能会把数据加载到数据库，也可能是在程序中使用。统一的注释字符可以让用户跳过不需要处理的头部信息。

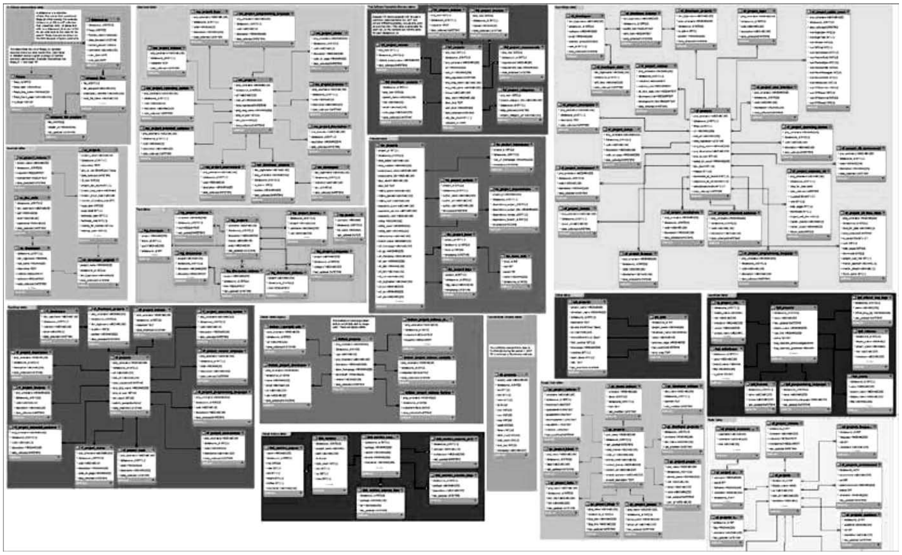
8.2.3　数据模型和图表

每当你要发布构建数据库用的SQL文件，或者是提供在线数据库查询访问的时候，使用可视化图表会对用户有很大的帮助，比如实体关系图（Entity-Relationship Diagram，ERD）。

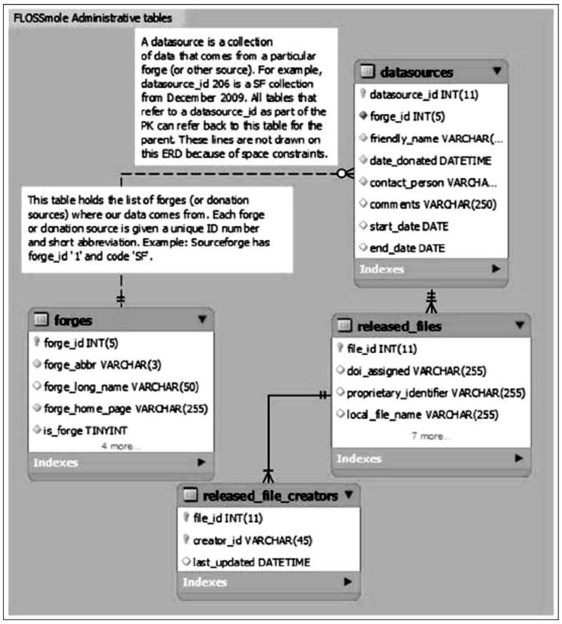
在我做过的一些项目中，我喜欢为表格提供文字性描述信息，就像上一节提到的文件头或是README文件，但同时还要提供一张可视化图表，用来表现数据表之间的关系。由于发布出来的数据库非常大，所以我在图表里涂上了各种鲜艳的颜色，并为其中的每一个部分加上不同的标注以表明这是数据库的哪个部分。

下面的截图[①]显示的是其中一张表现概要信息的大图。这里显示的是经过缩小处理后的实体关系图。

① 彩色图片请到www.ituring.com.cn/book/1702下载。——编者注



由于这个实体关系图非常大，看起来很困难，即使是在一个超大的显示器中，我也不得不采用不同的颜色来标记数据库的不同部分，并在有需要的地方添加上注释。下面的截图[①]是左上方橙色部分的近景展现。



数据库其中一个部分的近景图，包括数据表的相关注释信息

①彩色图片请到www.ituring.com.cn/book/1702下载。——编者注

通过这张图，用户可以对数据库的不同部分有一个大概的了解。更重要的是，注释信息也直接包含在图表中，如果用户想要查看某个特定的字段，可以参考README文件或者是对应文件内部的头信息。

许多RDBMS工具都可以用来创建ERD文件，MySQL Workbench就是其中之一（刚刚看到的带颜色的图表，就是我用这个工具创建的）。其他比较流行的工具还包括Microsoft Visio、Sparx Enterprise Architect和draw.io。而且许多这样的工具都可以连接到RDBMS系统，通过逆向工程操作从数据库中生成图表，或者是正向工程从图表中生成SQL语句。但无论是哪种情况，实体关系图都可以帮助用户更好地了解数据模型。

8.2.4　维基或 CMS

另一种组织项目文档的方式就是把它们发布到维基或是*内容管理系统*（Content Management System，CMS）上。可供我们选择的CMS和维基软件包有几百种，其中名气较大的有MediaWiki、WordPress、Joomla!和Drupal。此外，GitHub对托管在它上面的项目也提供了维基服务，类似的软件托管服务还有Sourceforge和Bitbucket。

你可以使用CMS或维基提供文件的下载链接，也可以使用CMS来发布文档与产品说明。我就曾在工作中用CMS存放实时更新的博客文章、示例插图和数据图表，甚至还把它当作文件仓库使用，用来存放对用户有帮助的数据脚本程序。

下面是数据项目的CMS或维基应该包含的一些常见文档内容。

- **项目说明**——告诉用户当前数据项目的目的以及项目负责人。这里还可以包含参与项目的建议，有关加入邮件列表或讨论区的方法，以及项目负责人的联系方式。
- **数据获取**——解释不同的数据访问方式。常见的方式包括直接访问数据库、文件下载或使用API。同时还解释了注册与登录的操作流程。
- **数据使用**——包括查询、使用示例、数据的图形化展现、图表和实体关系图等。同时还要提供都有哪些人使用了这些数据。另外，在必要的时候，列出你所期望的数据使用途径和许可条款。

在这一节中，我们讨论了多种与数据有关的文档编写方法，包括README文件、文件头、实体关系图和基于网络的解决方案。在这些讨论中，我们还涉及一些与数据许可协议有关的概念，另外还解答了一些数据引用与分享的问题。在下一节里，我们将深入讨论数据集许可协议的内容。

8.3　为数据设置使用条款与许可协议

数据发布计划的一个重要部分就是决定如何让用户使用、分析或是重新编排发布出来的数据。而这些对用户应该如何使用数据的期望，就是*数据使用条款*（Terms of Use，ToU）。另外，

数据使用条款还可以用来赋予用户一些具体权利，比如允许用户修改数据或者是重新发布数据。赋予用户的这些权利的集合称为*数据许可协议*。用户可以选择同意（或不同意）数据使用条款来使用（或不使用）你的数据集。他们也可以根据数据的许可证条款来决定是否要使用数据。

在这一节中，我们将概述一些可在设置用户的数据使用规范时作为参考的选项。同时我们还会了解一些需要在使用条款中放置的常见内容，以及可以直接应用到要发布数据集上的预设许可协议。

常见使用条款

并非所有人分享数据的目的都是一样的，以我个人为例，我参与的一个项目的具体目标就是为科学社区收集、清洗与重新发布数据。因为我本人是一名大学教授，所以我的部分工作职责就是发表学术研究论文、软件和一些对他人有用的数据集。因此，人们如何援引我的论文和数据集，对我来说是非常重要的。但是，一位非学术界的朋友就常常以匿名身份发布数据集，毫不在乎那些数据使用条款和注意事项。

下面列出的是在设置数据使用条款时需要考虑的一些内容。

- *引用*。当人们基于你的数据成果发布他们自己的内容时，你是否需要他们明确声明这些原始数据的来源？如果是的话，他们应该使用什么样的URL或引证呢？
- *隐私*。你为用户或用户信息设置了隐私保护条例吗？你需要用户遵守隐私或研究规章吗？例如，有的人会要求用户遵守他们院校研究委员会（IRB）或是其他研究伦理群体（如互联网研究员协会，AOIR）所规定的规章条款。
- *数据的合理使用*。你是否怀疑数据集可能会被乱用？会有人把数据从相应的环境中分离出来吗？这些数据是否会以不好的方式与其他数据结合使用？对于有些项目来说，最好还是为用户设置一些数据使用建议。
- *联系方式*。当用户想要使用你的数据时，你希望他们用什么方式联系你呢？他们是否有必要通知你呢？作为数据提供者，准备好一份关于如何联系以及是否有必要联系你的指南，对于那些对数据有疑问的用户是非常有帮助的。

正如我们之前在8.2节中讨论的那样，为潜在用户准备的数据集使用条款可以放在README文件、文件头或是网站上。如果你提供了在线数据库访问功能，那么可以通知用户，接受数据库系统的用户名和密码就等同于同意了你的条款。类似的做法也可以用在API的访问上，用户主动使用认证令牌或访问证书就表明同意了你设置的使用条款。

当然，这些做法都要符合不同国家和组织所定制的法律法规。当然，要是完全不借助外部的帮助，想把这些事情全都搞定也几乎是不可能的。为了帮助数据提供者为他们的用户设置使用条款，我们可以使用一些现成的通用许可协议。接下来我们就其中的两项展开讨论。

1. 知识共享许可协议

知识共享（Creative Commons，CC）许可协议是一些通用的条款集合，用于帮助有版权的材料的提供者保护他们的版权。这些许可协议规定了作品使用者应当如何使用这些来自原创作者的成果。通过事先声明的许可协议，作品的所有者可以避免为每个需要修改或重新发布作品的个人单独发放许可。

不过知识共享许可协议也是有缺陷的——但对你来说也许根本就不算回事儿，因为这完全取决于你如何使用它——知识共享许可协议是面向有版权的作品的。那你的数据库或数据集是否有版权呢？你有兴趣为数据库中的内容或者是数据库本身加上许可条款吗？要回答这些问题，你可以看看知识共享许可协议的维基文档，它会深入地讨论我们需要考虑的每一处细节。就连与数据和数据库有关的常见问题也都为我们一一解答了，访问地址为https://wiki.creativecommons.org/Data。

2. 开放数据库协议和开发数据共用协议

还有一种为数据添加许可条款的方案就是使用*开放数据库协议*（Open Database License，ODbL）。这个许可协议一般用于数据库。在*开放知识基金会*（Open Knowledge Foundation，OKF）上有一个两分钟长的用户指南，借此我们可以学习如何开放我们的数据，具体内容可以访问http://OpenDataCommons.org/guide/。

如果你还希望选择更多的方案，可以访问开放知识基金会的另一个网站http://OpenDefinition.org，这里有更多现成的许可协议。从非常开放的公共领域，到演绎作品的归属与共享，各种许可协议一应俱全。此外，他们还提供了一个开放数据手册，利用这个手册你可以方便地完成针对数据库或数据集的知识产权构建过程。你可以把开放数据手册下载下来使用，或者是直接访问它的在线地址：http://OpenDataHandbook.org。

8.4　数据发布

当你准备好数据包，就可以开启发布流程了。通过发布数据，可以让更多的人来使用它们。如果你已经对数据的用户团体有了预先判断，那么发布过程就跟往邮件列表或某个研究小组发送一个URL那么简单。但是，有时候我们创建的数据集可能会吸引更多、更复杂的受众群体。

8.4.1　数据集清单列表

互联网上有许多可用的数据集合列表，它们大多都是围绕某个主题来进行组织的。这些元数据集合（集合的集合）的发布者通常都非常乐于分享新的数据源。这些元数据集合可能会包含以下内容。

- 具有相同主题的数据集，例如，音乐数据、生物学数据或是新闻故事的文章集合。
- 与解决同种问题相关的数据集，例如，可以用于推荐系统开发的数据集，或是能够训练机器学习分类器的数据集。
- 与某个技术问题相关的数据集，例如，设计用于测试某个软件或硬件设计的数据集。
- 设计用于特定系统的数据集，例如，经过优化后可以用于学习R语言编程的数据集，或是用于Tableau可视化服务的数据集，又或是像Amazon Web Services这样基于云技术平台的数据集。
- 有着同种类型许可协议的数据集，例如，只适用于公共领域的数据，或是专门为学术研究而清洗过的数据。

如果你觉得上面的列表中没有体现出你的数据集的特点，或者是不满足你对元数据集合的要求，那么你完全可以创建一个适合自己使用的数据仓库。

8.4.2 Stack Exchange 上的 Open Data

从http://opendata.stackexchange.com可以访问Stack Exchange的Open Data讨论区，这里全是与开源数据集有关的问题和答案。我曾在这里找到过许多有趣的数据集，有时，我也会用我自己的数据集来回答其他人提出的问题。通过这个Q&A网站，我们还可以了解到人们都有哪些类型的问题，以及他们喜欢什么样的数据格式。

在Stack Exchange上，如果你想使用数据来回答其他人提出的问题，那就一定要保证数据的访问方式、相关文档和许可协议符合我们之前讨论过的标准。在Stack Exchange这样的网站上，这一点尤为重要，因为问题也好，答案也罢，用户都是有权为它们给出差评的。谁也不愿意将数据连同一大堆失效的链接和令人费解的文档一起发布出来。

8.4.3 编程马拉松

还有一种让人们积极参与数据活动的办法，就是把数据当作编程马拉松数据集发布出来。数据编程马拉松通常指的是，程序员和数据科学家们聚集在一起，花费一整天或者是几天的时间来操练各种不同的技能，或是解决与数据有关的某一类问题。

通过搜索引擎查询一下就可以得知当下编程马拉松所关心的内容。有的编程马拉松是由公司赞助发起的，而有的则是用于回应社会问题而举办的。这些活动中有很大一部分都有它们自己的维基文档，或是其他可以让你添加URL和数据描述的方法，借助这些途径，你可以发布在活动日要使用的数据集。其实还有一个不是特别好的方法可以推荐，那就是只举办一次编程马拉松，之后便逐渐地将该活动转换成其他形式。或者是不定时举办，并由一些临时成立的小组承办。

如果你的数据集是用于学术目的的，比如研究性质的数据集，你可能会考虑在学术会议期间

或之后，举办你自己的编程马拉松。这是一个让人们积极参与数据操作的绝佳方法，至少也是一个可以让你从参与人员中得到数据改善或构建建议的好机会。

8.5　小结

在这一章中，我们学习了分享数据成果的各种可行方法，讨论了数据打包与发布的解决方案和折中方法。另外还学习了文档编写的基础知识，包括用户需要知道的重要事情，以及如何在文档文件中把这些信息传达出去。在此过程中我们发现许可条款和使用约定几乎总是出现在文档中，但它们的真正含义是什么，又该如何根据自己的数据集情况进行选择呢？为了解答这些问题，我们又进一步了解了适用于数据项目的一些常见的使用条款，同时还有最为常见的许可方案：知识共享许可协议和开放数据库协议。最后，我们一起头脑风暴了实际中用得上的一些数据发布方法，包括数据的元数据集、Stack Exchange网站的Open Data讨论区，以及以数据为中心的编程马拉松。

到此为止，你应该已经对数据清洗项目有了全面的认识。在接下来的两章中，我们将完成更长、更具体的项目，并把之前学到的技术应用到其中。

第 9 章

Stack Overflow项目

接下来的两章会讲解两个完整的项目，我们会把之前学过的所有数据清洗技术付诸实践。我们可以把每个项目都想象成一次晚餐聚会，并可以借此机会在数据科学厨房里一显身手。为了成功举办晚餐聚会，我们应当事先准备好菜谱和客人的名单。但是，对于高手而言，如何应对计划之外的突发事件，这才算得上真正的考验。不管原来计划好的菜谱和购物清单有多么周详，有时还是有可能漏掉某些重要的原料。这种情况下我们是否可以随机应变，恰当地调整计划，解决碰到的各种挑战呢？

在这一章里，我们将使用来自Stack Overflow数据库的公开数据并进行数据清洗。Stack Overflow是问答网站Stack Exchange中的一部分。在这些网站上，提出好问题和回答问题可以赚取用户点数和徽章。在操练数据清洗技术过程中，我们会用到第1章中学到的六步处理方法。

- 确定问题种类——为什么要关注这种数据？
- 收集与保存数据，内容包括下载数据、提取Stack Overflow的转储数据、创建存放数据用的MySQL数据库，以及将数据导入MySQL数据库。由于Stack Overflow数据集很庞大，我们还得创建数据量较少的测试表，采用随机筛选数据的方法填充数据表。
- 在正式清洗整个数据集之前，在测试表上进行一些试验性的清洗工作。
- 分析数据。我们需要执行一些计算逻辑吗？应不应该利用聚合函数计算数据的个数或汇总值？是否需要按照某种方式对数据进行转换？
- 如果有可能的话，提供一些数据可视化实现。
- 调查研究并解决问题。我们的处理过程有效吗？最终取得成功了吗？

工作内容确实不少，不过我们准备得越早、越充分，数据科学晚餐聚会取得成功的机会就越大。

9.1 第一步：关于 Stack Overflow 的问题

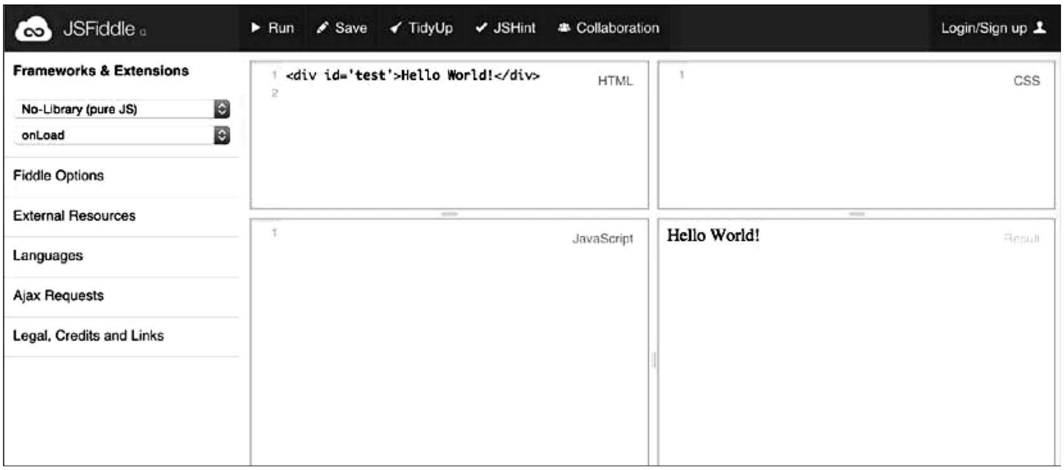
项目开始之前，我们需要提出一个只有经过数据分析才能回答的问题。那究竟应该从什么地方入手呢？首先，还是回顾一下Stack Overflow网站吧。众所周知，它是一个面向程序员的问答

网站，程序员们会在问题和答案中使用许多源代码、错误日志和配置文件。但有的时候，尤其是在Stack Overflow这样的大型网络平台上，个别人发布出来的文本数据会特别长，这就会让每一行数据在文本长度、数据格式和可读性方面的处理变得异常困难。

在阅读了许多含有大量文本内容的问题和答案之后，我想知道，在Stack Overflow上程序员是不是通过Pastebin或JSFiddle等外部网站来链接他们的代码或日志文件，比如你可以在http://www.Pastebin.com上粘贴大量的文本内容，既可以是程序源代码也可以是日志文件，之后你会得到一个可以与他人分享的简短URL。而且这些代码分享网站大多数都支持语法高亮显示，不过在默认情况下Stack Overflow并没有这样的功能。

在IRC和电子邮件中使用代码分享网站是一件十分常见的事情，那Stack Overflow上情况又如何呢？一方面，在IRC或电子邮件中使用链接可以让问题或答案的内容变得很简短，这样其余部分阅读起来就会容易得多。但另一方面，由于使用的代码分享网站不同，URL不见得会一直有效。这就意味着问题或答案可能会随着时间的推移因为链接的失效而失去了它们本身的价值。

此外，如果使用了JSFiddle，有可能会把问题搞得更复杂。在JSFiddle这样的*互动网站*上，你不仅可以把代码粘贴进去换取一个URL，还可以让其他人在浏览器中编辑与运行程序代码。这种方式非常适合Stack Overflow这样的提问与回答场景，尤其是像JavaScript这样基于浏览器的语言。然而，链接失效的问题依然存在。另外还有一点要注意的是，与Pastebin这样的代码分享网站相比，JSFiddle对于新手来说不是很容易上手。



JSFiddle上的四个窗口，分别对应HTML、CSS、JavaScript和运行结果

在Stack Overflow的社区讨论中，对于是否应该使用代码分享网站，以及包含代码分享网站链接、不包含代码的问题和答案的使用规则，有着诸多争论。大体上，即便人们认为使用代码分享网站很有帮助，他们也认识到保护Stack Overflow本身的长久性和有效性也是很重要的。社区的最后决定是，应当避免发布那些只提供链接而无代码的提问和回答。如果你对这些讨论有兴趣的话，请参考链接http://meta.stackexchange.com/questions/149890/。

我们讨论这个问题的目的并不是要选择站在哪一边，只是想提出几个简单的数据驱动问题。

(1) 在Stack Overflow上，人们使用Pastebin和JSFiddle（以及其他类似的代码分享网站）的频率是多少？

(2) 人们在问题和答案中使用代码分享网站的比率大吗？

(3) 含有代码分享网站URL的用户提交，是不是同时包含程序源代码？如果是的话，数量又有多少？

我们可以使用以上几个问题作为收集、存储和清洗Stack Overflow数据的目标。即使最后发现有的问题太难回答或者无法回答，牢牢记住这个整体目标也会对我们的清洗工作有着莫大的指导作用。并且，牢记这些问题能防止我们在实际工作中偏离问题方向，避免毫无意义的工作。

9.2　第二步：收集并存储 Stack Overflow 数据

在本书编写的时候，Stack Exchange为它旗下的所有网站产品都提供了数据下载服务，其中也包括来自Stack Overflow的数据，任何人都能够以XML文件格式免费下载它们。在这一节中，我们将下载Stack Overflow数据并把这些数据导入到MySQL数据库中。最后，我们还会创建几张数据量较小的表，专门供测试使用。

9.2.1　下载 Stack Overflow 数据

所有来自Stack Exchange的数据都可以在互联网档案馆（Internet Archive）上下载。在本书编写的时候，2014年9月的数据是当时最新的数据。每个Stack Exchange子站都有一个或多个这样的数据转储文件，并且每个文件都有它们自己对应的详细页面，访问地址是https://archive.org/details/stackexchange。

在众多的文件中，只有下面八个Stack Overflow文件才是我们需要的，它们按字母顺序排列。

stackoverflow.com-Badges.7z	77.8 MB
stackoverflow.com-Comments.7z	1.8 GB
stackoverflow.com-PostHistory.7z	9.4 GB
stackoverflow.com-PostLinks.7z	26.4 MB
stackoverflow.com-Posts.7z	5.7 GB
stackoverflow.com-Tags.7z	508.1 KB
stackoverflow.com-Users.7z	100.5 MB
stackoverflow.com-Votes.7z	385.0 MB

存放在Archive.org上的八个Stack Overflow文件

对于列表中的每一个文件，我们只需在下载链接上点击鼠标右键，就可以用浏览器把文件下载到本地磁盘。

9.2.2　文件解压

从下载下来的文件名中可以看出，它们都有一个以.7z结尾的文件扩展名。这是一种压缩文件格式。可以使用7-Zip软件或是与之兼容的软件包来解压。在第2章中我们讨论过常见的文件归档格式，但7-Zip并没有列位其中，你的电脑上也许也没有安装相关的兼容软件，如果真的是这样的话，那么我们就把这个当作第一个问题来解决吧。你可以先试着双击文件看看是否可以打开它，但是如果你从未安装过与.7z关联的软件，就需要安装一个适合的解压程序了。

- 对于Windows系统，你可以从7-Zip网站下载该软件：http://www.7-zip.org。
- 对于Mac OS X系统，你可以下载并安装Unarchiver，它是一个免费的工具，下载网站地址为http://unarchiver.c3.cx。

软件安装完成之后，按顺序分别解压每个文件。解压出来的文件会非常大，所以你得先确保磁盘有足够的空间来存放这些文件。

> 在我的操作系统上对比压缩版本与未压缩版本的数据文件大小时，结果显示未压缩版本几乎是压缩版本的十倍。而且在解压的时候每个文件都需要花费几分钟时间，当然，时间长度也跟使用的系统有关，所以请为这一步预留一些时间。

9.2.3　创建 MySQL 数据表并加载数据

现在拥有的这八个.xml文件，每一个都与即将要创建的数据表对应。要想创建这些表和数据库，可以利用phpMyAdmin或其他一些图形工具通过鼠标操作完成，或者是运行由Georgios Gousios编写的SQL完成，文件下载地址为https://gist.github.com/gousiosg/7600626。下载下来的代码中只包含了前六张表的`CREATE`和`LOAD INFILE`语句，因为另外两张表是在这个脚本编写之后才追加到转储数据库文件的。

要创建新表的结构，可以从Terminal窗口或shell中运行head命令来查看文件的前几行内容。下面的示例是从Terminal窗口中查看最小的文件PostLinks.xml：

```
head PostLinks.xml
```

前四行结果显示如下：

```
<?xml version="1.0" encoding="utf-8"?>
<postlinks>
  <row Id="19" CreationDate="2010-04-26T02:59:48.130" PostId="109"
  RelatedPostId="32412" LinkTypeId="1" />
  <row Id="37" CreationDate="2010-04-26T02:59:48.600" PostId="1970"
  RelatedPostId="617600" LinkTypeId="1" />
```

新的数据表中的每一行数据都与XML文件的<row>标签对应，标签中的每个属性都对应表中的一个字段。我们可以在文件Tags.xml上重新执行head命令来验证数据表中应该包含哪些字段。对于缺少的那两张数据表，可以通过下面的SQL代码中的CREATE语句和LOAD来创建：

```
CREATE TABLE post_links (
    Id INT NOT NULL PRIMARY KEY,
    CreationDate DATETIME DEFAULT NULL,
    PostId INT NOT NULL,
    RelatedPostId INT NOT NULL,
    LinkTypeId INT DEFAULT NULL
);

CREATE TABLE tags (
    Id INT NOT NULL PRIMARY KEY,
    TagName VARCHAR(50) DEFAULT NULL,
    Count INT DEFAULT NULL,
    ExcerptPostId INT DEFAULT NULL,
    WikiPostId INT DEFAULT NULL
);

LOAD XML LOCAL INFILE 'PostLinks.xml'
INTO TABLE post_links
ROWS IDENTIFIED BY '<row>';

LOAD XML LOCAL INFILE 'Tags.xml'
INTO TABLE tags
ROWS IDENTIFIED BY '<row>';
```

> 这里需要注意的是，LOAD XML语法得稍作调整，因为只有这样才可以访问存放在本地磁盘的数据文件。如果.xml文件存在本地计算机而非数据库服务器，那么你只需要在LOAD XML语句中添加单词LOCAL即可，具体做法请参照上面的示例代码，另外你也可以使用完整的文件路径。
>
> 关于MySQL的LOAD XML语法的更多信息请参考MySQL在线文档：http://dev.mysql.com/doc/refman/5.5/en/load-xml.html。

现在我们已经准备好了八张功能完备的数据表，并且每张都填满了数据。但是，这些表的数据量特别大。八张表的数据加起来竟然有一亿九千万条之多。另外还有一件事需要注意，那就是在数据清洗和数据分析前的准备过程中，如果我们在任何一张大表（如posts、comments、votes、post_history）中不小心犯下一个错误，就不得不重新创建这些数据表，而这个操作将花费很长时间。所以，在接下来的一步中，我们将学习如何创建测试用的数据表，这样就可以在程序或查询出错时对损失加以控制。

9.2.4　构建测试表

在这一节中，我们会针对原始数据表构建八个规模较小的数据表，之后再从原始表中随机选取一些数据进行填充。

第一步就是重新运行CREATE语句，但这次需要为表名添加test_前缀，下面是其中一张表的示例代码：

```
DROP TABLE IF EXISTS test_post_links;
CREATE TABLE test_post_links (
    Id INT NOT NULL PRIMARY KEY,
    CreationDate INT,
    PostId INT,
    RelatedPostId INT,
    LinkTypeId INT
);
```

除了表名前面的test_之外，这八张测试表与我们之前做的表没有任何区别。

接下来，我们需要把数据填充到新创建的测试表中。我们可以从原始表中选择前1000行数据放到测试表中。但是，这么做会有一个缺点，即由于原始数据是按照被插入Stack Overflow数据库的顺序排列的，结果就是这1000条样本数据在日期和时间上不会有太大不同。而我们实际希望的样本数据应该是随机发布的。那怎么才能随机选取这些数据呢？在此之前，我们从未解决过类似的问题，所以，这将是我们要在数据科学晚餐聚会上讨论的又一新话题。

随机抽取行数据的方法不止一个，但它们的效率还是有分别的。在这个项目里，效率还是比较重要的，因为我们要处理的数据非常庞大。本来是有一种随机抽取数据的方法的，那就是利用数字类型的主键字段Id，但可惜的是这些主键数字并不连续。行数据之间有许多缺口，例如在表post_links中，Id字段的前几个值分别是19、37、42和48。

实际上，这些数据缺口会对我们的随机抽取操作造成一定的影响，请看下面的操作步骤。

(1) 构建PHP脚本并运行下面的查询，找出表中最小和最大的Id值：

```
SELECT min(Id) FROM post_links;
SELECT max(Id) FROM post_links;
```

(2) 然后在同一个脚本中，在最小值和最大值之间生成随机数，并使用下面的语句找出随机数对应的行数据：

```
SELECT * FROM post_links WHERE Id = [random value];
```

(3) 重复上面的第二步，根据你自己的需要添加更多的数据。

但不走运的是，在处理Stack Overflow数据表时产生了许多失败的查询，post_links表就是个例子，这是因为字段Id里有许多数字缺口。举个例子来说，如果上面例子中的第二步生成的随机值是38会发生些什么呢？post_links表中没有Id为38的字段。这意味着我们需要及时发现这个错误，并改换另一个新的随机值。

> 每当这个时候，那些对SQL技术略知一二但不是非常了解的人，通常会建议在字段Id上使用MySQL的ORDER BY rand()，然后再执行LIMIT命令来限制我们需要的数据条数。但实际上这个方案也是有问题的，即使在排序时使用的是一个索引字段，ORDER BY rand()还是需要在每一行记录上执行一次，只有这样才能为每一行记录赋予一个新的随机值。所以，在像Stack Overflow这样的大型数据库上，这是行不通的。一次ORDER BY rand()查询会花费很长时间才能执行完毕。ORDER BY rand()对于小规模的数据表还是可以接受的，但不太适用于我们这个项目。

下面的PHP脚本程序才是最终的随机数据抽取解决方案，它向我们演示了如何为八张测试用的数据表分别填充1000条数据。并且在随机选取数据的时候都尽可能地做得恰到好处，避免在这个简单的问题上过度设计：

```
<?php //randomizer.php
// 每张测试用的数据表中应该存放的数据条数为多少?
$table_target_size = 1000;

// 连接到数据库，设置查询语句，运行查询语句
$dbc = mysqli_connect('localhost','username','password','stackoverflow')
       or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");

$tables = array("badges",
    "comments",
    "posts",
    "post_history",
    "post_links",
    "tags",
    "users",
    "votes");

foreach ($tables as $table)
{
```

```
    echo "\n=== Now working on $table ===\n";
      $select_table_info = "SELECT count(Id) as c, min(Id) as mn,
      max(Id) as mx FROM $table";
      $table_info = mysqli_query($dbc, $select_table_info);
      $table_stuff = mysqli_fetch_object($table_info);
      $table_count = $table_stuff->c;
      $table_min = $table_stuff->mn;
      $table_max = $table_stuff->mx;

      // 建立一个循环结构随机抓取一些数据并插入到新的数据表中
      $i=0;
      while($i < $table_target_size)
      {
          $r = rand($table_min, $table_max);
          echo "\nIteration $i: $r";
          $insert_rowx = "INSERT IGNORE INTO test_$table (SELECT *
          FROM $table WHERE Id = $r)";
          $current_row = mysqli_query($dbc, $insert_rowx);

          $select_current_count = "SELECT count(*) as rc FROM
          test_$table";
          $current_count= mysqli_query($dbc, $select_current_count);
          $row_count = mysqli_fetch_object($current_count)->rc;
          $i = $row_count;
      }
}
?>
```

在运行完这段代码之后，我们可以从结果中看出，这八张表中的数据才是我们需要的。有了这些小规模的数据表，我们的清洗练习可以进行得更顺利，犯错成本更低。如果在实际使用时发现需要更多的随机数据，只要加大变量`$table_target_size`的值并重新运行一次脚本就可以了。

构建测试表是一种好习惯，但前提是你知道应该怎么以简单有效的方法来创建它们。

9.3　第三步：数据清洗

还记得我们的任务目标吧，先分析问题、答案和用户评论中某些URL的引用频率，这当然要从Stack Overflow的`post`表和`comments`表开始才合理。但是，由于这些表的数据量比较大，所以我们实际要使用的是刚刚创建的表`test_posts`和表`test_comments`。如果稍后我们对查询结果有了信心，那就可以在大表上重新运行这些命令。

清洗任务与第7章的URL提取非常相似。但是，这个项目还有它自己的独特规则。

- 由于用户提交和用户评论属于不同的实体，所以我们应该为来自用户提交的URL（包括问题和答案）和来自用户评论的URL分别创建存储表。
- 每个问题、答案或用户评论都有可能包含多个URL信息。我们应该把这些URL全部保存

下来，并为URL的来源做好跟踪记录。

- 每个问题、答案或用户评论都有可能包含多份经过格式化的源代码。Stack Overflow提交内容里的程序源代码都是采用`<code>`标签单独隔离出来的。从用户提交的内容中分离出程序源代码，可以帮助我们回答有关代码分享URL和程序源代码共存的问题。如果有的话，一个链接中通常有多少代码呢？

> 从技术上讲，所有的用户提交在创建时都可以去除`<code>`标签，但通常有人会快速修正提交内容，引入这些有用的标签，并且会因此获得Stack Overflow用户点数。为了简化目标，我们假定这个项目中的所有代码都放在`<code>`标签中。

- 根据Stack Overflow数据库的文档（地址为http://meta.stackexchange.com/questions/2677/），实际上有八种类型的用户提交，问题和答案只是其中的两种。所以，在查询的时候，我们需要分别使用`postTypeId=1`和`postTypeId=2`把用户提交类型限定在问题和答案这两种类型中。
- 为了确保只从那些与问题和答案有关的用户评论中提取URL，暂不考虑其他类型的用户提交，我们需要结合用户提交表做一个联合查询，把结果限制在`postTypeId=1`或`postTypeId=2`范围之内。

9.3.1 创建新的数据表

利用下面的SQL语句，我们可以创建用于存放URL的数据表：

```
CREATE TABLE clean_comments_urls (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    commentId INT NOT NULL,
    url VARCHAR(255) NOT NULL
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS clean_posts_urls (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    postId INT NOT NULL,
    url VARCHAR(255) NOT NULL
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
```

另外，我们还需要创建一张新的数据表来存放从用户提交内容中删除的程序代码：

```
CREATE TABLE clean_posts_code (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    postId INT NOT NULL,
    code TEXT NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```


到此为止我们已经创建了三张数据表，其中有两张用来存放清洗后的URL，另外一张存放

清洗后的程序源代码。在下一个小节中，我们将把抽取出来的URL和源代码插入到这些新的数据表中。

9.3.2　提取 URL 并填写新数据表

我们可以修改在第7章中编写的脚本，用它抽取Stack Overflow场景下的URL，代码如下：

```
<?php // urlExtractor.php
// 连接到数据库
$dbc = mysqli_connect('localhost', 'username', 'password', 'stackoverflow')
       or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");

// 提取与提交信息有关的文本数据
// postTypeId=1 (问题)
// 或postTypeId=2 (答案)
$post_query = "SELECT Id, Body
    FROM test_posts
    WHERE postTypeId=1 OR postTypeId=2";

$comment_query = "SELECT tc.Id, tc.Text
    FROM test_comments tc
    INNER JOIN posts p ON tc.postId = p.Id
    WHERE p.postTypeId=1 OR p.postTypeId=2";

$post_result = mysqli_query($dbc, $post_query);
// 如果查询失败则终止处理
if (!$post_result)
    die ("post SELECT failed! [$post_query]" .  mysqli_error());

// 提取URL信息
$urls = array();
$pattern  = '#\b(([\w]+://?|www[.])[^\s()<>]+(?:\([\w\d]+\)|([^[:punct:]\s]|/)))#';

while($row = mysqli_fetch_array($post_result))
{
    echo "\nworking on post: " . $row["id"];
    if (preg_match_all(
        $pattern,
        $row["Body"],
        $urls
    ))
    {
        foreach ($urls[0] as $url)
        {
          $url = mysqli_escape_string($dbc, $url);
            echo "\n----url: ".$url;
            $post_insert = "INSERT INTO clean_posts_urls (id, postid, url)
                VALUES (NULL," . $row["Id"] . ",'$url')";
            echo "\n$post_insert";
            $post_insert_result = mysqli_query($dbc,
```

```
            $post_insert);
        }
    }
}

$comment_result = mysqli_query($dbc, $comment_query);
// 如果查询失败则终止处理
if (!$comment_result)
    die ("comment SELECT failed! [$comment_query]" .
    mysqli_error());

while($row = mysqli_fetch_array($comment_result))
{
    echo "\nworking on comment: " . $row["id"];
    if (preg_match_all(
        $pattern,
        $row["Text"],
        $urls
    ))
    {
        foreach ($urls[0] as $url)
        {
            echo "\n----url: ".$url;
            $comment_insert = "INSERT INTO clean_comments_urls
            (id, commentid, url)
                VALUES (NULL," . $row["Id"] . ",'$url')";
            echo "\n$comment_insert";
            $comment_insert_result = mysqli_query($dbc,
            $comment_insert);
        }
    }
}
?>
```

现在我们已经完成了数据表clean_post_urls和clean_comment_urls的填充。在我的测试表中，跑完上面的脚本之后只生成了大约100条用户评论URL和700条用户提交URL。当然，这对于正式开始之前的测试来说已经足够了。

9.3.3 提取代码并填写新表

要想提取嵌套在<code>标签内的文本数据并填写新表clean_posts_code，可以运行下面的脚本程序。这段程序与URL提取比较相似，但不需要搜索用户评论内容，因为用户评论中没有<code>标签修饰的代码。

9

在我的随机测试表中，第一条SELECT语句从总行数为1000的数据表test_post中筛选出约800行数据。但是，由于每条用户提交数据都可能包含多个程序代码片段，所以最终实际表中的记录超过了2000条。下面就是从<code>标签中提取嵌套代码的PHP脚本：

```
<?php // codeExtractor.php
// 连接到数据库
$dbc = mysqli_connect('localhost', 'username', 'password', 'stackoverflow')
    or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");

// 提取与提交信息有关的文本数据
// postTypeId=1 (问题)
// 或postTypeId=2 (答案)
$code_query = "SELECT Id, Body
    FROM test_posts
    WHERE postTypeId=1 OR postTypeId=2
    AND Body LIKE '%<code>%'";

$code_result = mysqli_query($dbc, $code_query);
// 查询失败则终止处理
if (!$code_result)
    die ("SELECT failed! [$code_query]" .  mysqli_error());

// 从提交数据中提取代码片段
$codesnippets = array();
$pattern  = '/<code>(.*?)<\/code>/';

while($row = mysqli_fetch_array($code_result))
{
    echo "\nworking on post: " . $row["Id"];
    if (preg_match_all(
        $pattern,
        $row["Body"],
        $codesnippets
    ))
    {
      $i=0;
        foreach ($codesnippets[0] as $code)
        {
            $code = mysqli_escape_string($dbc, $code);
            $code_insert = "INSERT INTO clean_posts_code (id, postid, code)
                VALUES (NULL," . $row["Id"] . ",'$code')";
            $code_insert_result = mysqli_query($dbc, $code_insert);
            if (!$code_insert_result)
                die ("INSERT failed! [$code_insert]" .
                mysqli_error());
            $i++;
        }
        if($i>0)
        {
            echo "\n Found $i snippets";
        }
    }
}
?>
```

现在我们已经将每条用户提交里包含的程序源代码提取出来了，并且也按照预期的设计把它

们存放到数据表clean_post_code里。

9.4　第四步：数据分析

在这一节，我们要编写一些代码来回答本章开头所提出的三个问题。具体是要找出：

- 在用户提交和用户评论中，使用到的不同类型代码分享URL的个数；
- 代码分享URL在问题和答案中的数量分别有多少；
- 在所有使用了代码分享URL的用户提交中，<code>的使用比率。

9.4.1　哪些代码分享网站最为流行

要想回答第一个问题，我们需要生成一份JSON格式的数据，其中应当包含代码分享网站URL及其使用次数，这些信息分别来自数据表clean_posts_urls和clean_comments_urls。通过这个简单的分析，我们可以找出在Stack Overflow数据文件中有哪些代码分享网站比较流行。在下面的PHP脚本中，程序会根据我们在数组变量$pastebins中预先列出的代码分享网站，从用户提交和用户评论的内容中统计出与之匹配的数据个数。由于脚本使用的是测试表，所以实际产生的数字会远远小于在真实表中运行出来的结果：

```
<?php // q1.php
// 连接到数据库
$dbc = mysqli_connect('localhost', 'username', 'password',
  'stackoverflow')
    or die('Error connecting to database!' . mysqli_error());
$dbc->set_charset("utf8");

// 下面的URL地址是我们需要查找与统计的内容
$pastebins = array("pastebin",
    "jsfiddle",
    "gists",
    "jsbin",
    "dpaste",
    "pastie");
$pastebin_counts = array();

foreach ($pastebins as $pastebin)
{
    $url_query = "SELECT count(id) AS cp,
          (SELECT count(id)
          FROM clean_comments_urls
          WHERE url LIKE '%$pastebin%') AS cc
        FROM clean_posts_urls
        WHERE url LIKE '%$pastebin%'";
    $query = mysqli_query($dbc, $url_query);
    if (!$query)
        die ("SELECT failed! [$url_query]" .  mysqli_error());
```

9

```
    $result = mysqli_fetch_object($query);
    $countp = $result->cp;
    $countc = $result->cc;
    $sum = $countp + $countc;

    array_push($pastebin_counts, array('bin' => $pastebin,
                                    'count' => $sum));
}
// 在最终的列表转换成json格式之前，需要对数据进行排序处理
// 按照数据条数进行排序，由高到低
foreach ($pastebin_counts as $key => $row)
{
    $first[$key]  = $row['bin'];
    $second[$key] = $row['count'];
}

array_multisort($second, SORT_DESC, $pastebin_counts);
echo json_encode($pastebin_counts);
?>
```

从脚本输出的结果中，我们可以找到利用测试表得出的JSON数据内容。下面是来自我的测试数据表的统计信息：

```
[{"bin":"jsfiddle","count":44},{"bin":"jsbin","count":4},{"bin":"paste
bin","count":3},{"bin":"dpaste","count":0},{"bin":"gists","count":0},{
"bin":"pastie","count":0}]
```

> 你的实际运行结果值可能会有所不同，这是因为我们随机选择的数据是不同的。

在9.5节中，我们会使用这份JSON数据构建一个条形图表。但在此之前，还是让我们先回答之前提出的另外两个问题。

9.4.2　问题和答案中的代码分享网站都有哪些

第二个问题是，问题和答案中，代码分享网站的使用比率哪个更高。要解答这个问题，我们需要运行一系列SQL查询。第一个查询要找出的是数据表`clean_posts_urls`中每种类型的用户提交的个数，包括问题和答案：

```
SELECT tp.postTypeId, COUNT(cpu.id)
FROM test_posts tp
INNER JOIN clean_posts_urls cpu ON tp.Id = cpu.postid
GROUP BY 1;
```

从我的随机测试表得出的结果是，一共有237个问题和440个答案。

postTypeId	COUNT(cpu.id)
1	237
2	440

在phpMyAdmin中显示的问题URL个数和答案URL个数

现在，我们需要知道的是：在问题和答案共同组成的677个URL中，每种类型的代码分享URL对应的数据量到底有多少？这个问题可以通过下面的SQL语句找到答案：

```
SELECT  tp.postTypeId, count(cpu.id)
FROM test_posts tp
INNER JOIN clean_posts_urls cpu ON tp.Id = cpu.postId
WHERE cpu.url LIKE '%jsfiddle%'
OR cpu.url LIKE '%jsbin%'
OR cpu.url LIKE '%pastebin%'
OR cpu.url LIKE '%dpaste%'
OR cpu.url LIKE '%gist%'
OR cpu.url LIKE '%pastie%'
GROUP BY 1;
```

下面的表格显示的就是查询结果。引用代码分享网站的问题有18个，引用代码分享网站的答案有24个。

postTypeId	count(cpu.id)
1	18
2	24

在phpMyAdmin中，问题和答案中分别包含的URL引用个数

有一点需要记住，上面的查询会对每个出现的URL都做一次计算。所以，如果某个`postId`引用了五个URL，那最后的统计结果得经过五次计算才能得出。如果想要知道有多少用户提交一次或多次引用了一个URL，我们需要把上面的两个查询都做一下调整。下面是调整后的查询语句，用来统计URL表里非重复用户提交的个数：

```
SELECT tp.postTypeId, COUNT(DISTINCT cpu.postId)
FROM test_posts tp
INNER JOIN clean_posts_urls cpu ON tp.Id = cpu.postId
GROUP BY 1;
```

下面的截图显示了包含URL的问题和答案的个数。

postTypeId	COUNT(DISTINCT cpu.postid)
1	81
2	222

在phpMyAdmin中显示的包含URL的问题和答案

利用下面的查询语句，我们可以计算出URL表中包含某个代码分享网站的用户提交个数：

```
SELECT  tp.postTypeId, count(DISTINCT cpu.postId)
FROM test_posts tp
INNER JOIN clean_posts_urls cpu ON tp.Id = cpu.postId
WHERE cpu.url LIKE '%jsfiddle%'
OR cpu.url LIKE '%jsbin%'
OR cpu.url LIKE '%pastebin%'
OR cpu.url LIKE '%dpaste%'
OR cpu.url LIKE '%gist%'
OR cpu.url LIKE '%pastie%'
GROUP BY 1;
```

查询结果如下，与我们预期的一样，数量变少了。在测试数据集中，包含代码分享URL的问题有11个，答案有16个。合算起来，一共有37条用户提交至少引用了代码分享URL一次。

postTypeId	count(distinct cpu.postid)
1	11
2	16

在phpMyAdmin中，包含代码分享URL的问题和答案统计数量

虽然从测试结果中可以看出，人们在答案里使用的代码分享URL数量要比问题中多，但我们还是需要使用真实数据表中全部的问题和答案重新再做一次比较。另外，我们还应该以百分比的形式分别计算出问题和答案所对应的比例。如果把总数纳入考虑范围内容，我们就可以这样说："如果只考虑那些包含URL的问题和答案，在81条问题中有11条使用了代码分享URL（13.6%），在222条答案中有16条使用了代码分享URL（7.2%）。"与此类似，在你的实际结果中，问题中使用的代码分享网站的比例几乎是答案的两倍。

不论是在哪个数据分析项目中，到了这个时候你都应该准备好更多的问题，例如：

- 随着时间的变化，问题和答案中的代码分享URL的使用率会如何变化？
- 在受欢迎程度的投票上，包含代码分享URL的问题表现如何？
- 使用代码分享URL提问的用户都有什么样的特点？

由于这本书是谈论数据清洗的，而且到现在为止还没有可用的可视化数据，所以我先不回答这些外围问题。另外，对于之前提出的三个问题，还剩下最后一个要回答，之后我们再考虑怎么

对数据进行可视化操作。

9.4.3　提交内容会同时包含代码分享 URL 和程序源代码吗

要回答第三个问题，需要比较Stack Overflow上的问题和答案中包含的代码数量，特别要关注那些包含由`<code>`标签分隔的源代码的提交。在9.3节中，我们已经从测试表中提取了用户提交里的代码，并创建了一张新表存放这些代码片段。现在，通过下面的查询语句可以计算出包含程序源代码的用户提交数量：

```
SELECT count(DISTINCT postid)
FROM clean_posts_code;
```

在我的测试数据集中，查询结果显示在1000条测试用的用户提交中，包含程序源代码的用户提交数量为664个。换一种表述方式就是：在1000条用户提交中，有664条记录至少包含一个`<code>`标签。

想要计算出有多少条用户提交同时包含程序源代码和代码分享URL，可以使用下面的SQL查询语句：

```
SELECT count(DISTINCT cpc.postid)
FROM clean_posts_code cpc
INNER JOIN clean_posts_urls cpu
ON cpu.postId = cpc.postId;
```

我的测试结果显示一共有175条这样的数据。这意味着在1000条用户提交中，有17.5%同时包含程序源代码和URL。

现在，为了找出同时包含程序源代码和代码分享URL的用户提交个数，我们可以使用下面的SQL查询语句进一步缩小结果范围：

```
SELECT count(DISTINCT cpc.postid)
FROM clean_posts_code cpc
INNER JOIN clean_posts_urls cpu
ON cpu.postId = cpc.postId
WHERE cpu.url LIKE '%jsfiddle%'
OR cpu.url LIKE '%jsbin%'
OR cpu.url LIKE '%pastebin%'
OR cpu.url LIKE '%dpaste%'
OR cpu.url LIKE '%gist%'
OR cpu.url LIKE '%pastie%';
```

从结果数据中我们可以看出，其实只有25条用户提交既包含程序源代码又包含代码分享URL。而且从第二个问题的答案中我们已经知道，一共有37个不重复的用户提交（包括问题和答案）至少使用过一次这种代码分享网站。所以25/37就是大约68%。如果在更大的数据集上运行这些查询，在看到结果那一刻，你会越发觉得有成就感。

接下来要做的是对前面的结果数据做一些简单的数据可视化处理，只有这样我们才算是真正地完成了数据科学的六步过程。

9.5　第五步：数据可视化

数据可视化就好比晚餐聚会中的饭后甜点。每个人都喜欢内容丰富的图形化展现，因为它们非常直观。但这本书的重点是数据清洗，而非数据分析和可视化，所以这里我们要实现的可视化图表是很简单的。在后面的程序代码中，我们将使用JavaScript的可视化库D3来以图表的形式显示第一个问题的结果数据。这个可视化实现比我们在第4章使用D3实现的可视化要简单得多。我们可以试着回想一下当时构建的网络关系图的复杂程度，而这次一个简简单单的条形图就足够表现我们需要的标签和统计数据了。

下面就是可视化实现需要使用的HTML和JavaScript/D3代码。这份代码是对Mike Bostock编写的*Let's Build a Bar Graph*教程的扩展，教程地址为http://bl.ocks.org/mbostock/3885304。其中一个扩展点就是读取先前在q1.php脚本中生成的JSON文件。由于之前的JSON文件格式非常清晰，而且已经经过由高到低的排序处理，所以条形图的构建也变得轻松了：

```
<!DOCTYPE html>
<meta charset="utf-8">
<!--
this code is modeled on mbostock's
"Let's Make a Bar Chart" D3 tutorial
available at http://bl.ocks.org/mbostock/3885304
My modifications:
* formatting for space
* colors
* y axis labels
* changed variable names to match our data
* loads data via JSON rather than .tsv file
-->
<style>
.bar {fill: lightgrey;}
.bar:hover {fill: lightblue;}
.axis {font: 10px sans-serif;}
.axis path, .axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
.x.axis path {display: none;}
</style>
<body>
<script src="d3.min.js"></script>
<script>

var margin = {top: 20, right: 20, bottom: 30, left: 40},
```

```
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var x = d3.scale.ordinal()
    .rangeRoundBands([0, width], .1);

var y = d3.scale.linear()
    .range([height, 0]);

var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left");

var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

d3.json("bincounter.php", function(error, json)
{
    data = json;
    draw(data);
});

function draw(data)
{
    x.domain(data.map(function(d) { return d.bin; }));
    y.domain([0, d3.max(data, function(d) { return d.count; })]);

    svg.append("g")
        .attr("class", "x axis")
        .attr("transform", "translate(0," + height + ")")
        .call(xAxis);

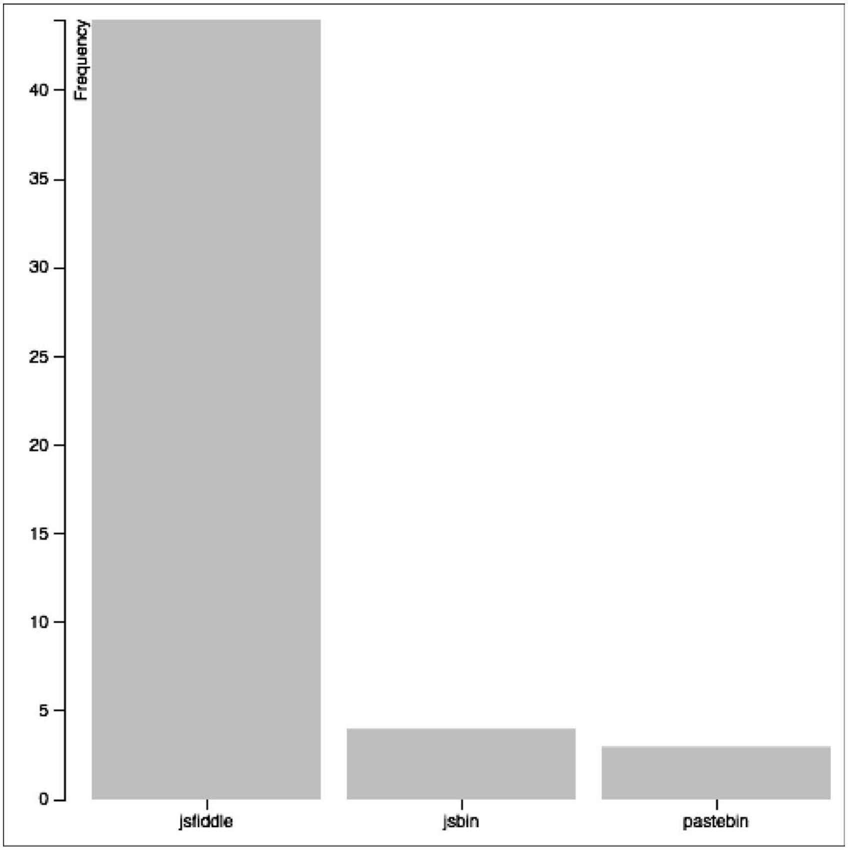
    svg.append("g")
        .attr("class", "y axis")
        .call(yAxis)
      .append("text")
        .attr("transform", "rotate(-90)")
        .attr("y", 6)
        .attr("dy", ".71em")
        .style("text-anchor", "end")
        .text("Frequency");

    svg.selectAll(".bar")
        .data(data)
      .enter().append("rect")
        .attr("class", "bar")
        .attr("x", function(d) { return x(d.bin) ; })
```

```
        .attr("width", x.rangeBand())
        .attr("y", function(d) { return y(d.count); })
        .attr("height", function(d) { return height - y(d.count); });
}

</script>
</body>
</html>
```

我们可以把这段内容保存为名为q1chart.html的文件，然后在浏览器中打开。代码执行过程中会调用脚本q1.php，并利用脚本生成的JSON文件在D3中构建成图表，下面是图表左侧部分的截图。



D3可视化图表的JSON格式URL统计信息

从条形图表中可以看出，指向JSFiddle的URL是最多的，至少在我的测试数据集中是这样。虽然从q1.php生成的JSON数据中就已经看出这个结果，但图表看起来还是更方便一些。在下一节中，我们将对所有结果和操作做一个总结，并继续讨论这个项目的下一个目标。

9.6 第六步：问题解析

根据9.4和9.5节中的查询结果和可视化图表，我们现在可以回答项目最初提出的三个问题了。

第一个问题是要找出用户提交和用户评论中使用到的代码分享网站数量的不同。从脚本q1.php和数据可视化条形图可以得知，JSFiddle是六种代码分享URL里使用最频繁的。

第二个问题是在问题和答案两者之间，哪一个使用代码分享URL的频率更高。查询结果表明，问题中使用代码分享URL的比率几乎是答案中的两倍，但由于我们使用的是测试数据集，所以测试数据量比较小一些。

第三个问题是要查出人们是否听从了Stack Overflow的建议，在使用代码分享URL的同时还附加了程序源代码。在前面的测试数据集中，查询结果显示有25条用户提交（总数为37）既包括代码分享URL，又包括源代码。建议的接受比率为68%。

除了上面的问题之外，我们还可以提出更多的问题来进行分析解答，也可以使用更多的方法来扩展这个实验，使它更加有趣。但是现在，我们应该把项目的存储和清洗步骤转移到完整的数据集上。

9.7 从测试表转向完整数据表

在项目开始的前期，我们生成了一些只含有1000条数据的小规模测试表，以便在毫无压力的环境中开发项目。有了这些数据量可控的小规模数据表，我们可以放心地校验查询语句的正确性，并对联合查询、子查询、正则表达式等内容进行反复实验。如果我们对之前编写的查询语句和脚本程序的结果都比较满意的话，那么现在就可以重新梳理我们的执行步骤，完成对全数据量数据表的操作了。

下面是将项目转移到完整数据表的步骤。

(1) `DROP`测试表：

```
DROP TABLE IF EXISTS test_badges;
DROP TABLE IF EXISTS test_comments;
DROP TABLE IF EXISTS test_posts;
DROP TABLE IF EXISTS test_post_history;
DROP TABLE IF EXISTS test_post_links;
DROP TABLE IF EXISTS test_tags;
DROP TABLE IF EXISTS test_users;
DROP TABLE IF EXISTS test_votes;
```

9

(2) 清空数据表`cleaned_posts_code`、`cleaned_posts_urls`和`cleaned_ comments_ urls`：

```
TRUNCATE TABLE cleaned_posts_code;
TRUNCATE TABLE cleaned_posts_urls;
TRUNCATE TABLE cleaned_comments_urls;
```

(3) 编辑脚本urlExtractor.php和codeExtractor.php，将SELECT语句中的用户提交表从test_posts表改到全数据量表。具体做法可以参考下面的代码：

```
SELECT Id, Body FROM posts
```

(4) 重新运行脚本urlExtractor.php和codeExtractor.php，再次用清洗过的代码和URL填充刚刚清空（执行过truncate操作）的数据表。

至此，我们已经拿到了可以用于分析和可视化实现的干净的代码数据表和URL数据表。在执行这些操作的时候不要着急，因为其中的许多查询和脚本都是比较耗时的。一方面是用户提交表的数据量很大，另一方面是许多针对文本字段的查询语句里使用了大量的模糊查询技术。

9.8　小结

在这个项目中，我们提出了在Stack Overflow上代码分享URL使用程度的问题，这类网站如http://www.Pastebin.com和http://www.JSFiddle.net。为了回答与之相关的问题，我们先是从Stack Exchange公共文件发布网站上下载了Stack Overflow的用户提交数据和一些其他的Stack Overflow数据。之后创建了一个MySQL数据库和八张数据表来存放这些数据。接着又创建了测试用的数据表，每张表的数据量为1000，这些数据都是随机选择的。从测试数据表中我们提取了每个问题、答案和用户评论中使用的URL，并把它们保存在新的干净的数据表中。我们也把问题和答案中使用到的程序源代码单独存放到新的数据表中。最后，我们构建了一些简单的查询语句并实现了数据可视化，利用这些结果回答了项目开始之初提出的几个问题。

尽管结果平淡无奇，但从数据清洗角度来看，我们的晚餐聚会还是很成功的。我们成功地制定了计划，并有条不紊地应用到实践中，然后根据实际需要做了相应的调整。现在该是进入最后一个项目的时候了，那将是一个完全不同的晚餐聚会菜单。

在下一章，我们将从著名的Twitter数据中收集并清洗属于我们自己的数据集。

第 10 章 Twitter项目

与第9章一样，这一章描述的是另一场晚餐聚会。在整个数据科学过程的每一个阶段，我们将演示不同的数据清洗技术。前一章的项目中使用的数据来自Stack Overflow，清洗时使用了MySQL和PHP技术，可视化实现则是依靠JavaScript的D3库完成的。而在这一章，我们要使用的数据来自Twitter，并改用MySQL和Python完成数据的收集和清洗过程，最后会用JavaScript和D3实现数据的可视化。与前一个项目类似，这一章的项目也会采用相同的数据科学过程。

(1) 问题定位——为什么要关注这种数据？

(2) 数据的收集与保存，内容包括数据下载，提取对公共开放的推文ID数据集，以及使用程序重新下载原始推文数据。这一步也需要创建测试用的小规模数据表。

(3) 在提取与保存我们需要的数据的同时，完成数据清洗任务。在这一步中我们会编写一个用于数据加载的Python程序，只提取有用的字段，并把它们写入测试数据表中。

(4) 分析数据。我们需要执行一些计算逻辑吗？应不应该利用聚合函数计算数据的个数或汇总值？是否需要按照某种方式对数据进行转换？

(5) 如果有可能的话，提供一些数据可视化实现。

(6) 调查研究并解决问题。我们的处理过程有效吗？最终取得预期的效果了吗？

和前一个项目一样，大部分工作都集中在数据的收集、存储和清洗上。在这个项目中，我们将密切关注不同时期的数据，从最初的文本文件，到后来的JSON文件，再到最后的MySQL数据库。每种格式的数据都源自不同的收集或清洗过程，并且每次得到的结果都会被传递到下一个步骤中使用。通过这种形式我们了解到，数据的收集、存储和清洗操作都是迭代进行的——一个操作的输出结果可以是另一个操作的输入数据——并不仅仅是简单的线性执行。

10.1 第一步：关于推文归档数据的问题

Twitter是一个流行的微博平台，全世界有好几百万人都在用它分享对时事的观点或看法。由于Twitter的发布和阅读比较容易，尤其是在移动设备上，它俨然已经成为分享有关政治危机和抗

议等公共事件的信息，或追踪热点问题的重要平台。被保存下来的推文数据就好像是一种时空胶囊，反映了大事件发生时公众的情绪。而且，被“定格”下来的推文本身不会受到记忆差错或事态变化的影响。学者和媒体专家们可以收集并研究这些与时事相关的推文归档数据，以此来了解更多的与事件有关的公众舆论，或是信息的传播方式，甚至是事件的细节、发生时间和起因。

现在许多人都开始公开他们的推文数据集。下面是一些可供公众下载的推文归档数据。

- 2014年8月10日至27日，在美国密苏里州的弗格森事件后，与抗议和动乱有关的推文内容。其访问地址为https://archive.org/details/ferguson-tweet-ids。
- 在2011年阿拉伯之春事件中，发自利比亚、巴林、埃及等国家的推文内容。其访问地址为http://dfreelon.org/2012/02/11/arab-spring-twitter-data-now-available-sort-of/。
- 2014年5月到6月，与标签#YesAllWomen和#NotAllMen有关的推文。其访问地址为http://digital.library.unt.edu/ark:/67531/metadc304853/。

下一步，我们需要选择其中的一个主题来下载对应的数据，并利用这些数据展开后续工作。因为弗格森事件推文是上面三个数据集中最新最完整的，所以我选择它作为实验对象。然而，无论你选用的是哪个数据集，这里讨论的概念和基本步骤都是通用的。

在本章中我们提出的问题很简单：当人们在推文中谈论弗格森事件时，提到的最多的互联网域名有哪些？与第9章的问题集相比，这个问题非常简单，但是这里的数据集处理方式有所不同，所以说一个简单的问题就足以让我们再举办一次数据科学晚餐聚会。

10.2　第二步：收集数据

与我们在第7章中使用的推文归档数据有所不同，这次使用的推文归档数据不再包含推文的正文内容。Twitter的服务条款在2014年曾经做过调整，随意发布他人推文正文内容的行为违背了该条款。在这次使用的新版本推文归档文件中，我们能找到的只有推文编号（ID）。真正的推文数据需要通过这些数据间接地收集。之后我们才可以进行推文数据存储和分析操作。还有一点需要注意，不论是在项目实施过程中还是完成之后，我们都不可以重新发布推文正文内容和它们的元数据信息，唯一可以发布的是编号信息。

> 对于研究人员来说，收集推文数据可能不太方便，但Twitter改变他们服务条款的目的是为了保护推文原作者的版权。尤其是在推文删除操作上，这一点更为重要了。如果某条推文信息在网络上被随意复制与发布，并由第三方以数据文件的形式保存下来，那么这条推文将永远无法被真正地删除。通过只允许研究人员收集推文的ID编号，Twitter可以保护推文原作者删除推文的权利。用一个已经删除的推文ID向服务器发出请求是不会得到结果的。

10.2.1　下载并提取弗格森事件的数据文件

与弗格森事件相关的推文数据可以从互联网档案馆（Internet Archive）上以压缩文件形式获取。只需使用浏览器访问地址https://archive.org/details/ferguson-tweet-ids，就可以下载一个体积为147 MB的ZIP文件。

然后可以通过下面的命令提取文件中的内容:

```
unzip ferguson-tweet-ids.zip
```

通过`ls`命令能显示目录ferguson-tweet-ids中的内容，其中含有两个压缩文件：ferguson-indictment-tweet-ids.zip和ferguson-tweet-ids.zip。现在唯一需要做的就是解压其中一个文件供项目使用，所以我选择了下面的文件：

```
unzip ferguson-tweet-ids.zip
```

文件解压之后会生成一些清单文件和一个数据文件夹。在数据文件夹内部有一个经过gzip压缩的文件。它的解压方法如下：

```
gunzip ids.txt.gz
```

该操作会生成一个名为ids.txt的文件。这正是我们需要的文件。接下来就是熟悉文件的内容。

运行`wc`命令可以查看文件中的数据规模。在命令提示符下运行`wc`命令后，我们可以看到文件的行数、单词个数和字符数的统计信息：

```
megan$ wc ids.txt
 13238863 13238863 251538397 ids.txt
```

第一个数字表明文件ids.txt中有多少行数据，总数高达1300万。接下来我们可以使用`head`命令看一下文件的内容：

```
megan$ head ids.txt
501064188211765249
501064196642340864
501064197632167936
501064196931330049
501064198005481472
501064198009655296
501064198059597824
501064198513000450
501064180468682752
501064199142117378
```

`head`命令显示了文件的前十行记录，从中我们可以看出每行都是一个由18位数字组成的推文ID。

10

10.2.2　创建一个测试用的文件

在这个阶段，我们要创建一个小规模的测试文件来完成后续的流程。之所以这么做，原因和第9章一样。因为原始文件实在是太大了，所以我们只使用一小部分数据就好，这样就可以把犯错成本限制在可控范围内。还有一点好处是，在测试程序代码的时候不需要等待太长时间来完成每一步。

与上一章的练习有所不同，测试数据是否要随机选择并不重要。从前面`head`命令显示的结果来看，数据并不是按照从小到大的顺序排列的。事实上，我们也没有办法了解原始数据到底是根据什么条件进行编排的。因此，只要抓取前1000条推文ID并把数据存放到另外一个文件就可以了。下面命令生成的文件就是我们要操作的测试数据集：

```
head -1000 ids.txt > ids_1000.txt
```

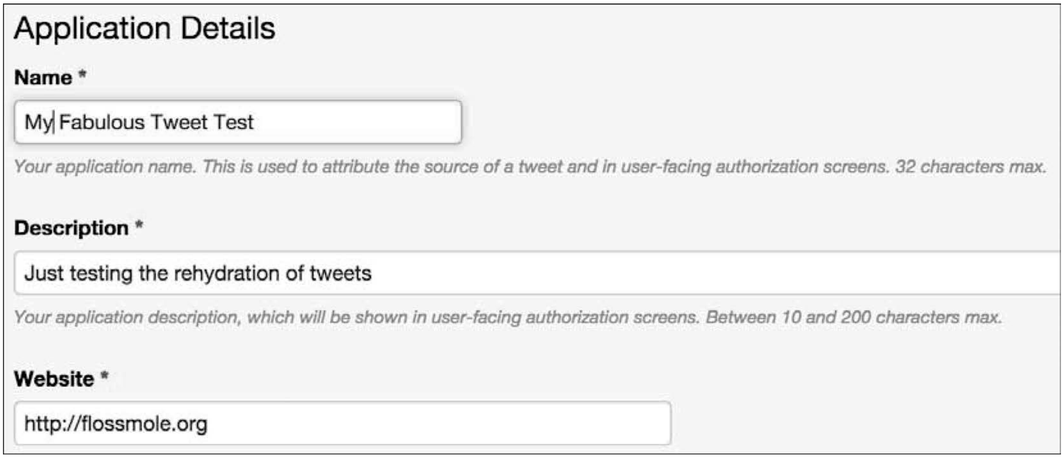
10.2.3　处理推文 ID

下面使用测试数据集里的1000条推文ID，根据这些编号信息收集原始推文的正文内容。为此，需要用到一个用Python编写的小工具twarc，该工具的作者是Ed Summers，也正是他把所有与弗格森事件有关的推文做了归档处理。我们要做的就是准备一份推文ID编号列表，然后利用Twitter API逐条地获取原始推文的正文内容。使用Twitter API的时候，必须先设置好Twitter开发者账号。所以，接下来要讲述的就是Twitter账号的准备，以及twarc的安装和使用。

1. 设置Twitter开发者账号

要想设置Twitter开发者账号，得先访问https://apps.twitter.com并用你的Twitter账号登录进去。如果你还没有Twitter账号，那就先申请一个，然后再继续后面的步骤。

使用Twitter账号登录进去后，点击http://apps.twitter.com页面上的Create New App。然后填写创建应用所需的详细信息（需要提供应用的名字，诸如My Tweet Test之类；一小段描述信息；一个URL，可以是非永久性的）。下面是我填写的应用创建表单，仅供参考：



含有示例数据的Create New App表单

勾选其中的复选框表明你同意开发者协议，然后重新回到Twitter应用的列表页面，这时刚刚创建的应用会出现在列表顶端。

接下来，为了使用应用，需要获取一些能让应用正常工作的关键信息。点击刚刚创建好的应用，你会从页面的上方看到四个页签。我们需要的数据处在名为Keys and Access Tokens的页签中。截图如下所示。



新创建的Twitter应用的页签界面

这个页面中有许多数字和保密码，但我们需要关注的只有四项：

- CONSUMER_KEY
- CONSUMER_SECRET
- ACCESS_TOKEN
- ACCESS_TOKEN_SECRET

不管你使用的是哪种Twitter API编程技术，至少在此时此刻，你一定需要这四项内容。twarc也好，别的工具也罢，都得遵守这个约定。这些信息可以让Twitter对你的身份进行验证并授予被请求资源的访问权限。

10

> 这些API认证信息也是Twitter限制请求发送次数和请求发生频率的依据。不过工具twarc可以替我们处理好这些事，所以我们不必对流量限制过分担心。要想了解更多关于Twitter流量限制的信息，请查询他们的开发者文档，地址为https://dev.twitter.com/rest/public/rate-limiting。

2. 安装twarc

既然已经取得了Twitter的认证信息，接下来就可以安装twarc了。

GitHub上twarc的下载页面是https://github.com/edsu/twarc。在这个页面上，有一份关于工具使用方法和相关选项的文档。

要想在Canopy Python环境下安装twarc，得先启动Canopy，然后从Tools菜单中选择Canopy Command Prompt。

然后在命令行中，输入下面的安装命令：

```
pip install twarc
```

这个命令会安装twarc，并可以让我们从命令行或是Python脚本程序中以命令方式调用它。

3. 运行twarc

现在我们可以从命令行中运行twarc来处理前面创建的测试文件ids_1000.txt。这次用到的命令非常长，因为我们需要传递之前在Twitter网站上创建的四个长长的密令。为了减少犯错机会，我先使用文本编辑器Text Wrangler来构建命令，然后再把它粘贴到命令提示符里去。最终构建好的命令应该与下面的例子一致，只是要把`abcd`替换成实际的密令或密钥：

```
twarc.py --consumer_key abcd --consumer_secret abcd --access_token
abcd --access_token_secret abcd --hydrate ids_1000.txt >
tweets_1000.json
```

注意，这条命令会把输出结果重定向到一个名为tweets_1000.json的JSON文件中。在文件内部，与ID对应的每条推文数据都以JSON格式展现。让我们看看新文件的长度：

```
wc tweets_1000.json
```

从工具`wc`的运行结果来看，文件一共才有894行，这表明有些推文信息没有找到（因为我最初放在数据集中的数据共有1000条）。如果在我编写本书之后又有一部分推文被删除的话，那么你的文件会比这个还要小。

让我们再通过下面的命令看看文件的内容：

```
less tweets_1000.json
```

当然，我们还可以在文本编辑器中打开文件。

JSON文件中的每一行记录都代表一条推文，内容样式与下面截图中的例子相似。但这个推文截图不是来自弗格森事件数据集，因为我无权发布与之相关的推文内容。所以，我使用了一条专为第2章讨论UTF-8编码创建的推文。因为这条推文是为本书创建的，我对其中的内容拥有所有权，所以我可以在不违反Twitter服务条款的情况下向读者展现相应的JSON格式数据。下面是这条推文在Twitter上的展现样式。



Titter网站上的推文样式

下面是该推文内容经过twarc处理后的JSON格式展现。为了更容易地了解到推文中都有哪些属性，我特意在每个JSON元素之间添加了换行符：

```
{"contributors": null,
"truncated": false,
"text": "Another test. \u00c9g elska g\u00f6gn. #datacleaning",
"in_reply_to_status_id": null,
"id": 542486101047275520,
"favorite_count": 0,
"source": "<a href=\"http://twitter.com\" rel=\"nofollow\">Twitter Web
Client</a>",
"retweeted": false,
"coordinates": null,
"entities":
{"symbols": [],
"user_mentions": [],
"hashtags":
[{"indices": [29, 42],
"text": "datacleaning"}],
"urls": []},
"in_reply_to_screen_name": null,
"id_str": "542486101047275520",
"retweet_count": 0,
"in_reply_to_user_id": null,
"favorited": false,
"user":
```

```
{"follow_request_sent": false,
"profile_use_background_image": false,
"profile_text_color": "333333",
"default_profile_image": false,
"id": 986601,
"profile_background_image_url_https": "https://pbs.twimg.com/profile_
background_images/772436819/b7f7b083e42c9150529fb13971a52528.png",
"verified": false,
"profile_location": null,
"profile_image_url_https": "https://pbs.twimg.com/profile_
images/3677035734/d8853be8c304729610991194846c49ba_normal.jpeg",
"profile_sidebar_fill_color": "F6F6F6",
"entities":
{"url":
{"urls":
[{"url": "http://t.co/dBQNKhR6jY",
"indices": [0, 22],
"expanded_url": "http://about.me/megansquire",
"display_url": "about.me/megansquire"}]},
"description": {"urls": []}},
"followers_count": 138,
"profile_sidebar_border_color": "FFFFFF",
"id_str": "986601",
"profile_background_color": "000000",
"listed_count": 6,
"is_translation_enabled": false,
"utc_offset": -14400,
"statuses_count": 376,
"description": "Open source data hound. Leader of the FLOSSmole
project. Professor of Computing Sciences at Elon University.",
"friends_count": 82,
"location": "Elon, NC",
"profile_link_color": "038543",
"profile_image_url": "http://pbs.twimg.com/profile_images/3677035734/
d8853be8c304729610991194846c49ba_normal.jpeg",
"following": false,
"geo_enabled": false,
"profile_banner_url": "https://pbs.twimg.com/profile_
banners/986601/1368894408",
"profile_background_image_url": "http://pbs.twimg.com/profile_
background_images/772436819/b7f7b083e42c9150529fb13971a52528.png",
"name": "megan squire",
"lang": "en",
"profile_background_tile": false,
"favourites_count": 64,
"screen_name": "MeganSquire0",
"notifications": false,
"url": "http://t.co/dBQNKhR6jY",
"created_at": "Mon Mar 12 05:01:55 +0000 2007",
"contributors_enabled": false,
"time_zone": "Eastern Time (US & Canada)",
"protected": false,
"default_profile": false,
"is_translator": false},
```

```
"geo": null,
"in_reply_to_user_id_str": null,
"lang": "is",
"created_at": "Wed Dec 10 01:09:00 +0000 2014",
"in_reply_to_status_id_str": null,
"place": null}
```


JSON对象不仅表现了推文的正文和发布的日期与时间，就连发布人的信息也包含在内。

> 单单一条推文的处理就产生了这么大的信息量，而我们实际要处理的可是1300万条。所以，当你准备在本章最后处理全部的弗格森事件数据集时，请务必记得这一点。

10.3　第三步：数据清洗

到这里，我们可以正式开始清洗JSON文件了，具体操作就是提取每条需要长期保存的推文细节信息。

10.3.1　创建数据表

由于我们一开始提出的问题就是针对URL的，所以在处理过程中只提取那些包含URL和推文ID的数据就足够了。除此之外，为了方便数据清洗以及和与前面第7章中的Sentiment140数据集做比较，我们还需要设计出数据规模较小的数据表。

- `tweet`表，用于放置推文信息
- `hashtag`表，用于放置推文内容中引用的标签信息
- `URL`表，用于放置推文内容中引用的URL信息
- `mentions`表，用于放置推文中包含的提及用户信息

这与之前在第7章中设计的方案比较类似，只是我们得自己动手从推文中解析出标签、URL和提及用户信息。但借助工具twarc，实际上我们已经节省了一些工作量。

下面是创建四张测试表用的`CREATE`语句：

```
CREATE TABLE IF NOT EXISTS ferguson_tweets (
  tid bigint(20) NOT NULL,
  ttext varchar(200) DEFAULT NULL,
  tcreated_at varchar(50) DEFAULT NULL,
  tuser bigint(20) DEFAULT NULL,
  PRIMARY KEY (tid)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS ferguson_tweets_hashtags (
  tid bigint(20) NOT NULL,
```

```
  ttag varchar(200) NOT NULL,
  PRIMARY KEY (tid, ttag)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS ferguson_tweets_mentions (
  tid bigint(20) NOT NULL,
  tuserid bigint(20) NOT NULL,
  tscreen varchar(100) DEFAULT NULL,
  tname varchar(100) DEFAULT NULL,
  PRIMARY KEY (tid,tuserid)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS ferguson_tweets_urls (
  tid bigint(20) NOT NULL,
  turl varchar(200) NOT NULL,
  texpanded varchar(255) DEFAULT NULL,
  tdisplay varchar(200) DEFAULT NULL,
  PRIMARY KEY (tid,turl)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

另外还有一件事情需要注意，存放推文数据的测试表是采用utf8mb4编码创建的。这是因为推文内容中可能会包含UTF-8编码范围之外的字符。实际上，推文中的一些特殊字符需要占用的存储空间，已经超过MySQL原生UTF-8字符集所能支持的三字节。因此，我们在设计核心的推文表时，采用MySQL的utf8mb4编码来存放数据，这种编码在MySQL 5.5和更高的版本中都支持。如果你的MySQL版本比较旧，或是出于某些其他原因不能使用utf8mb4编码，那么可以使用MySQL早期的UTF-8-general编码，但要小心处理那些由表情符号引起的编码错误。如果你恰巧在做`INSERT`操作时碰到这种错误，MySQL大多数时候会抛出1366错误，并在消息中给出incorrect string value字样。

现在我们已经创建完所有的测试表了，接下来就可以继续后面的数据筛选与加载工作了。

10.3.2　用 Python 为新表填充数据

我们需要编写Python脚本来加载JSON文件，提取那些有用的字段值，并为前面创建的四张测试表填充数据。另外还有一些额外的重要注意事项，我将一一介绍。

在运行这份脚本之前，我们需要安装好Python的`MySQLdb`模块。对于Canopy Python的用户来说，很容易从包管理器中安装这些模块。只需在Package Manager中搜索MySQLdb，然后点击安装即可：

```
#jsonTweetCleaner.py
import json
import MySQLdb

# 打开数据库连接
db = MySQLdb.connect(host="localhost",\
```

```
    user="username", \
    passwd="password", \
    db="ferguson", \
    use_unicode=True, \
    charset="utf8")
cursor = db.cursor()
cursor.execute('SET NAMES utf8mb4')
cursor.execute('SET CHARACTER SET utf8mb4')
cursor.execute('SET character_set_connection=utf8mb4')

# 打开json格式的推文数据
with open('tweets_1000.json') as f:
    for line in f:
        # 将每一条推文都放进字典中
        tweetdict = json.loads(line)

        # 获取每一条推文并写进数据库表中
        tid    = int(tweetdict['id'])
        ttext  = tweetdict['text']
        uttext = ttext.encode('utf8')
        tcreated_at = tweetdict['created_at']
        tuser  = int(tweetdict['user']['id'])

        try:
            cursor.execute(u"INSERT INTO ferguson_tweets(tid,
ttext, tcreated_at, tuser) VALUES (%s, %s, %s, %s)", \
(tid,uttext,tcreated_at,tuser))
            db.commit() # 在使用MySQLdb时，每次数据变化后必须提交更改内容
        except MySQLdb.Error as error:
            print(error)
            db.rollback()

        # 处理推文中涉及的标签
        hashdict = tweetdict['entities']['hashtags']
        for hash in hashdict:
            ttag = hash['text']
            try:
                cursor.execute(u"INSERT IGNORE INTO
ferguson_tweets_hashtags(tid, ttag) VALUES (%s, %s)",(tid,ttag))
                db.commit()
            except MySQLdb.Error as error:
                print(error)
                db.rollback()

        # 处理推文中涉及的URL
        urldict = tweetdict['entities']['urls']
        for url in urldict:
            turl      = url['url']
            texpanded = url['expanded_url']
            tdisplay  = url['display_url']

            try:
                cursor.execute(u"INSERT IGNORE INTO  ferguson_tweets_
urls(tid, turl, texpanded, tdisplay)
```

```
VALUES (%s, %s, %s, %s)", (tid,turl,texpanded,tdisplay))
                db.commit()
            except MySQLdb.Error as error:
                print(error)
                db.rollback()

        # 处理推文中涉及的用户
        userdict = tweetdict['entities']['user_mentions']
        for mention in userdict:
            tuserid = mention['id']
            tscreen = mention['screen_name']
            tname   = mention['name']

            try:
                cursor.execute(u"INSERT IGNORE INTO
ferguson_tweets_mentions(tid, tuserid, tscreen, tname)
VALUES (%s, %s, %s, %s)", (tid,tuserid,tscreen,tname))
                db.commit()
            except MySQLdb.Error as error:
                print(error)
# 断开服务器连接
db.close()
```



如果你想详细了解JSON格式中每一个推文字段的信息，请参见Twitter的API文档。在从JSON数据中提取字段值的时候，文档中Users、Entities和Entities in Tweets这几节内容对我们有着重要的指导性作用。相关的文档信息可以从https://dev.twitter.com/overview/api/找到。

脚本运行完成之后，四张测试表中都会填满数据。当时在我的MySQL服务器中，运行完ids_1000.txt之后推文表中共有893行记录，标签表有1048条记录，提及用户表有896条记录，URL表有371条记录。如果你的记录少于这些，请检查一下是不是因为有的推文已经被删除。

10.4　第四步：简单的数据分析

假设我们想要了解在弗格森事件数据集中哪些域名被引用的次数最多。要回答这个问题，只需从数据表`ferguson_tweets_urls`的`tdisplay`字段中，提取数据中所引用的URL域名。在实际操作过程中，URL地址中第一个斜线（/）之前的内容就是我们要考虑的。

利用下面的SQL查询可以获取域名数据并计算出推文引用该域名的次数：

```
SELECT left(tdisplay,locate('/',tdisplay)-1) as url,
  count(tid) as num
FROM ferguson_tweets_urls
GROUP BY 1 ORDER BY 2 DESC;
```

最后的查询结果与下面的示例数据相似（以1000条记录为测试样本）：

url	num
bit.ly	47
wp.me	32
dlvr.it	18
huff.to	13
usat.ly	9
ijreview.com	8
latimes.com	7
gu.com	7
ift.tt	7

上面的数据集片段只显示了前几行内容，实际的结果中还有一些使用短域名服务生成的特殊内容，比如`bit.ly`。在这些特殊的短域名中还有一部分是来自Twitter自己网站的，如`t.co`，在查询语句里改用字段`display`就可以把它们移除。

在下一节中，我们可以使用这些统计数字以与第9章类似的方式构建一张条形图。

10.5 第五步：数据可视化

要想构建一张D3图，需要按照第9章中的流程，编写一个用于数据库查询的PHP脚本，然后再使用JavaScript把数据结果实时地输出到条形图中去。另外还有一种办法就是用Python生成CSV文件，然后利用结果文件生成图表。因为在前一章里我们已经使用过PHP方案，所以在这里我们改用CSV方案。这也可以让本章所使用的程序语言风格保持一致。

通过下面的脚本程序我们可以连接到目标数据库，并选出最常用的15个URL和它们对应的引用次数，然后再把这些信息写到CSV文件里：

```
import csv
import MySQLdb

# 打开数据库连接
db = MySQLdb.connect(host="localhost",
    user="username",
    passwd="password",
    db="ferguson",
    use_unicode=True,
    charset="utf8")

cursor = db.cursor()
cursor.execute('SELECT left(tdisplay, LOCATE(\'/\',
    tdisplay)-1) as url, COUNT(tid) as num
    FROM ferguson_tweets_urls
    GROUP BY 1 ORDER BY 2 DESC LIMIT 15')

with open('fergusonURLcounts.tsv', 'wb') as fout:
```

```
    writer = csv.writer(fout)
    writer.writerow([ i[0] for i in cursor.description ])
    writer.writerows(cursor.fetchall())
```

CSV文件生成之后，就可以把它传入D3条形图模版中，看看会有什么样的效果吧。下面是文件buildBarGraph.html中的内容：

> 请确保D3库和CSV文件都保存在本地目录，这一点与我们在前一章的处置方式一样。

```
<!DOCTYPE html>
<meta charset="utf-8">
<!--
this code is modeled on mbostock's
"Let's Make a Bar Chart" D3 tutorial
available at http://bl.ocks.org/mbostock/3885304
My modifications:
* formatting for space
* colors
* y axis label moved
* changed variable names to match our data
* loads data via CSV rather than TSV file
-->

<style>
.bar {fill: lightgrey;}
.bar:hover {fill: lightblue;}
.axis {font: 10px sans-serif;}
.axis path, .axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
.x.axis path {display: none;}
</style>
<body>
<script src="d3.min.js"></script>
<script>

var margin = {top: 20, right: 20, bottom: 30, left: 40},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var x = d3.scale.ordinal()
    .rangeRoundBands([0, width], .1);

var y = d3.scale.linear()
    .range([height, 0]);

var xAxis = d3.svg.axis()
    .scale(x)
```

```
    .orient("bottom");

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left");

var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top +
")");

d3.csv("fergusonURLcounts.csv", type, function(error, data) {
  x.domain(data.map(function(d) { return d.url; }));
  y.domain([0, d3.max(data, function(d) { return d.num; })]);

  svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + height + ")")
      .call(xAxis);

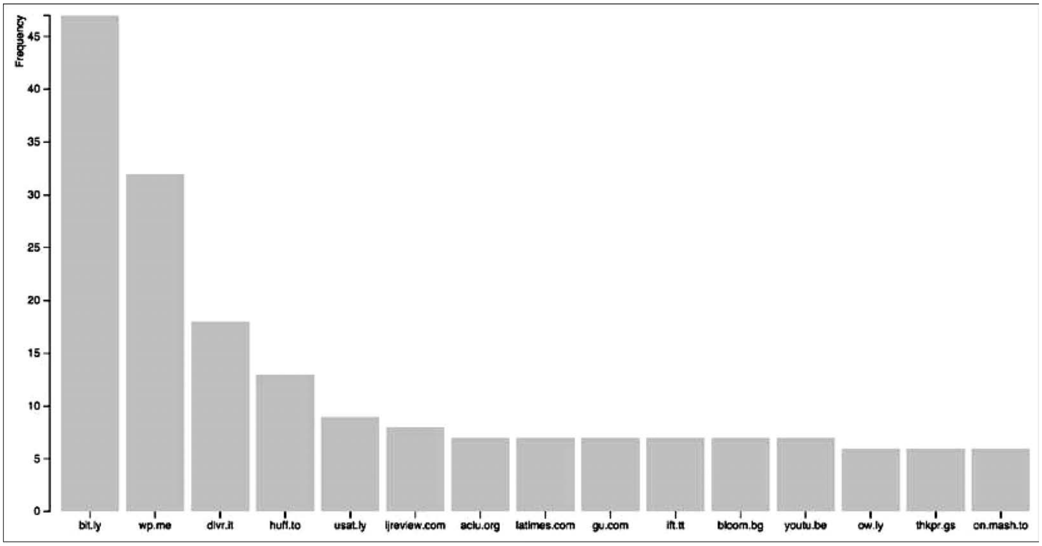
  svg.append("g")
      .attr("class", "y axis")
      .call(yAxis)
    .append("text")
      .attr("transform", "rotate(-90)")
      .attr("y", 6)
      .attr("dy", "-3em")
      .style("text-anchor", "end")
      .text("Frequency");

  svg.selectAll(".bar")
      .data(data)
    .enter().append("rect")
      .attr("class", "bar")
      .attr("x", function(d) { return x(d.url) ; })
      .attr("width", x.rangeBand())
      .attr("y", function(d) { return y(d.num); })
      .attr("height", function(d) { return height - y(d.num); });
});

function type(d) {
  d.num = +d.num;
  return d;
}

</script>
</body>
</html>
```

最终生成的条形图显示如下。请记住，我们使用的是测试数据集，所以数字看起来很小。



利用CSV数据绘制出来的D3条形图

10.6　第六步：问题解析

由于数据可视化不是本书的主要目标，所以我们没有必要在图表的复杂程度上下太多工夫，只要能够说明通过弗格森事件数据集还可以发掘出很多更有趣的样式，而不仅仅是能找出最常用的URL，这样就足够了。既然你已经学会了下载和清洗这些杂乱的数据集，或许你可以继续发挥一下自己的想象力，去发现一些有趣的样式了。只是在公开数据成果的时候，请记得不要随意发布推文正文或相关的元数据信息。不过发布推文ID或是其中的一个子集倒是可以的，前提是这些信息对于你的问题确实有用。

10.7　把处理过程应用到全数据量（非测试用）数据表

就像在第9章中那样，我们创建了一些测试用的数据表，这样我们的项目就可以在有限数据量的条件下进行。当你准备收集全部的推文内容的时候，请先做好心理准备，因为这个过程需要花费相当长一段时间。Twitter会限制用户的访问速率，这就是twarc会运行很长时间的原因。Ed Summers在他的博文中说明，采集弗格森事件的推文信息几乎花费了一周时间，这篇博文的地址为http://inkdroid.org/journal/2014/11/18/on-forgetting/。当然，如果你做了细心的准备，那么运行一次就可以得到你想要的结果了。

另外还有一种可以加快推文ID处理的办法，那就是找一位小伙伴合作。将推文ID文件一分为二，每人负责处理分给自己的那一份。在数据清洗过程中，请务必确保两人的数据处理结果都`INSERT`到同一张数据表。

下面是将1000条测试数据改为完整数据集的修改步骤。

(1) 清空数据表ferguson_tweets、ferguson_tweets_hashtags、ferguson_tweets_mentions和ferguson_tweets_urls：

```
TRUNCATE TABLE ferguson_tweets;
TRUNCATE TABLE ferguson_tweets_hashtags;
TRUNCATE TABLE ferguson_tweets_mentions;
TRUNCATE TABLE ferguson_tweets_urls;
```

(2) 使用完整版本的ids.txt文件替代之前使用的ids_1000.txt文件，再次运行twarc：

```
twarc.py --consumer_key abcd --consumer_secret abcd --access_token
abcd --access_token_secret abcd --hydrate ids.txt > tweets.json
```

(3) 重新运行Python脚本jsonTweetCleaner.py。

全部操作完成之后，数据库中的数据都已经清洗完毕，其中的推文正文、标签、提及用户和URL都可以直接用于数据分析和可视化实现。由于每张表中的记录都比较多，所以在可视化实现时花费的时间可能会比较长，具体时长取决于查询语句的执行时间。

10.8 小结

在这个项目中，我们学会了如何根据推文编号重新构建完整的推文列表。我们先是下载了一些符合Twitter服务条款的推文数据，并把这些归档数据拆分成可以用于测试的小规模数据表。之后学习了怎么把每条推文转换成JSON格式的数据，这当中使用到了Twitter API和命令行工具twarc。然后又学习了如何使用Python从JSON格式的数据中提取需要的数据，并把结果保存到MySQL数据库中。最后运行了一些查询语句统计出使用最为频繁的URL的个数，并用D3绘制了一张条形图。

在本书里，我们已经了解了各种数据清洗任务，包括简单的任务和复杂的任务。一路走下来，为了应对不同的工作，许多不同的编程语言、工具和技术轮番上阵。我希望你在学习更多新技能的同时，能够巩固与完善已经掌握的技能。

到此，我们最后一次晚餐聚会也完满结束了，你现在已经完全有能力在物资齐全、干净整洁的数据科学厨房开启属于你自己的项目了。那到底应该从何开始呢？

- 喜欢竞赛和奖品吗？Kaggle经常会在他们的网站上举办数据分析比赛，网站地址为http://kaggle.com。你可以以个人名义参与，或是作为团队成员参与。有许多团队都需要干净的数据，所以这是一种很好的参与方式。
- 如果你更热衷于公共服务，我猜想School of Data可能会比较适合你。他们的网站是http://schoolofdata.org，在这里开办的*Data Expeditions*课程常常会吸引世界各地的专家和

业余爱好者一同来解决现实生活中与数据有关的问题。

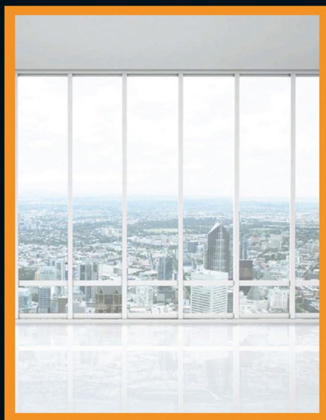
- 为了扩大数据清洗练习的范围，我建议你亲自动手处理一些对公众开放的数据。KDnuggets上有一些很好的公众数据列表，其中一些列表还包含二级列表，访问地址为http://www.kdnuggets.com/datasets/。
- 你喜欢像第9章这样的例子吗？位于http://meta.stackexchange.com上的Meta Stack Exchange是一个专门讨论Stack Exchange工作原理的网站。用户常常在这儿讨论如何查询Stack Overflow数据和这些数据的使用方式。或许你也可以在这里提出那些与Stack Overflow有关的数据清洗问题。另外，Stack Exchange还有其他一些与数据清洗有关的下属网站。其中一个便是Open Data Stack Exchange，访问地址为http://opendata.stackexchange.com。
- 时下Twitter数据是非常流行的。如果你喜欢处理Twitter数据的话，可以考虑把本章的项目再提升一个级别，根据公众开放的推文数据提出问题并给出答案。比如收集并甄选出属于你自己的推文ID集合？如果你根据某个兴趣主题构建了一份干净的推文ID集合并把研究成果发布出来，研究人员和其他数据科学家一定会非常感激的。

祝你在数据清洗的旅途上一路顺风，扬帆起航吧！

数据清洗是数据挖掘与分析过程中不可缺少的一个环节，但因为数据类型极其复杂，传统的清洗脏数据工作单调乏味且异常辛苦。如果能利用正确的工具和方法，就可以让数据清洗工作事半功倍。

本书从文件格式、数据类型、字符编码等基本概念讲起，通过真实的示例，探讨如何提取和清洗关系型数据库、网页文件和PDF文档中的数据。最后提供了两个真实的项目，让读者将所有数据清洗技术付诸实践，完成整个数据科学过程。

如果你是一位数据科学家，或者从事数据科学工作，哪怕是位新手，只要对数据清洗有兴趣，那么本书就适合你阅读！



- 理解数据清洗在整个数据科学过程中的作用
- 掌握数据清洗的基础知识，包括文件清洗、数据类型、字符编码等
- 发掘电子表格和文本编辑器中与数据组织和操作相关的重要功能
- 学会常见数据格式的相互转换，如JSON、CSV和一些特殊用途的格式
- 采用三种策略来解析和清洗HTML文件中的数据
- 揭开PDF文档的秘密，提取需要的数据
- 借助一系列解决方案来清洗存放在关系型数据库里的坏数据
- 创建自己的干净数据集，为其打包、添加授权许可并与他人共享
- 使用书中的工具以及Twitter和Stack Overflow数据，完成两个真实的项目

[PACKT] PUBLISHING

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/数据科学

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-42047-3

定价：49.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks